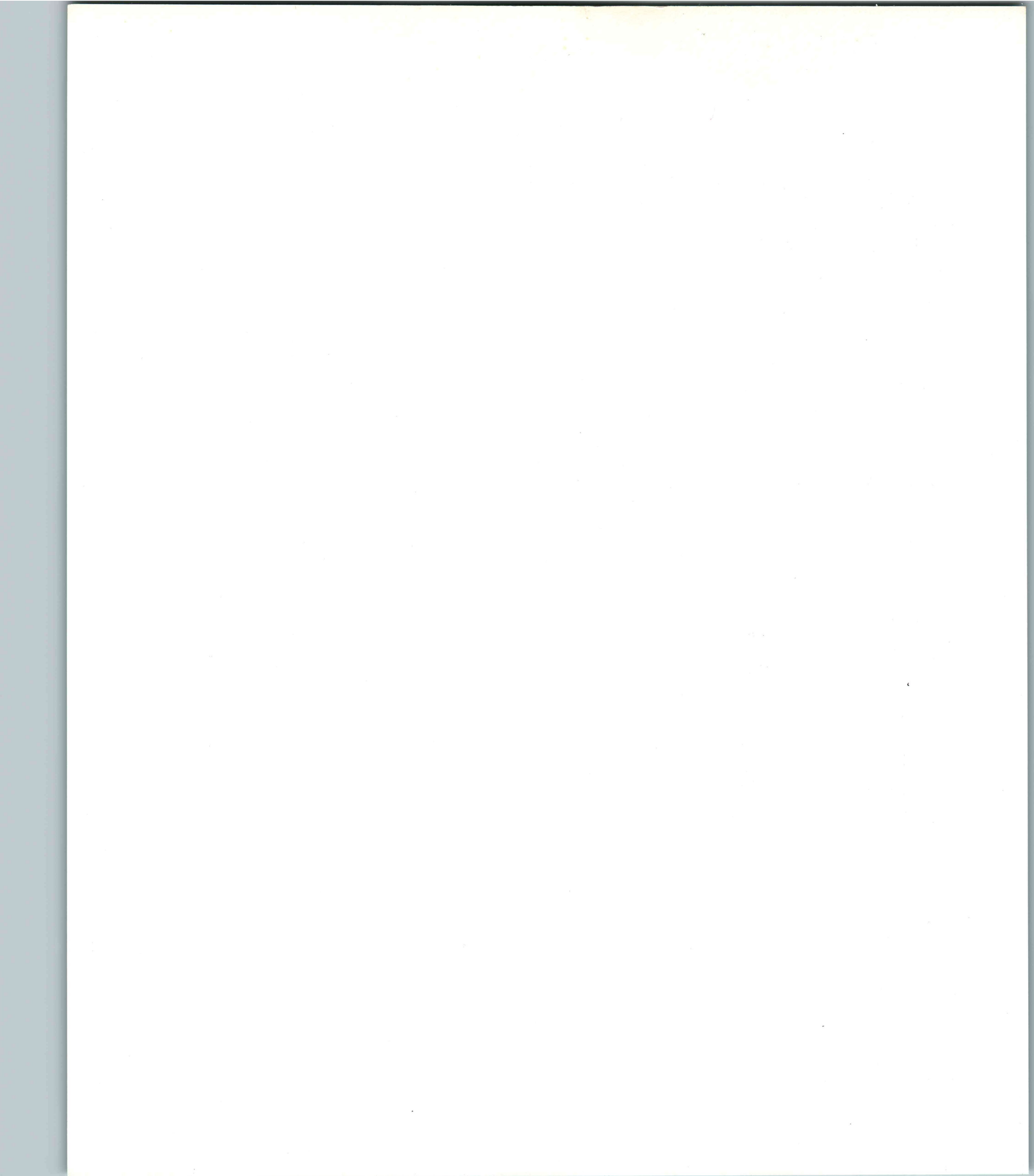
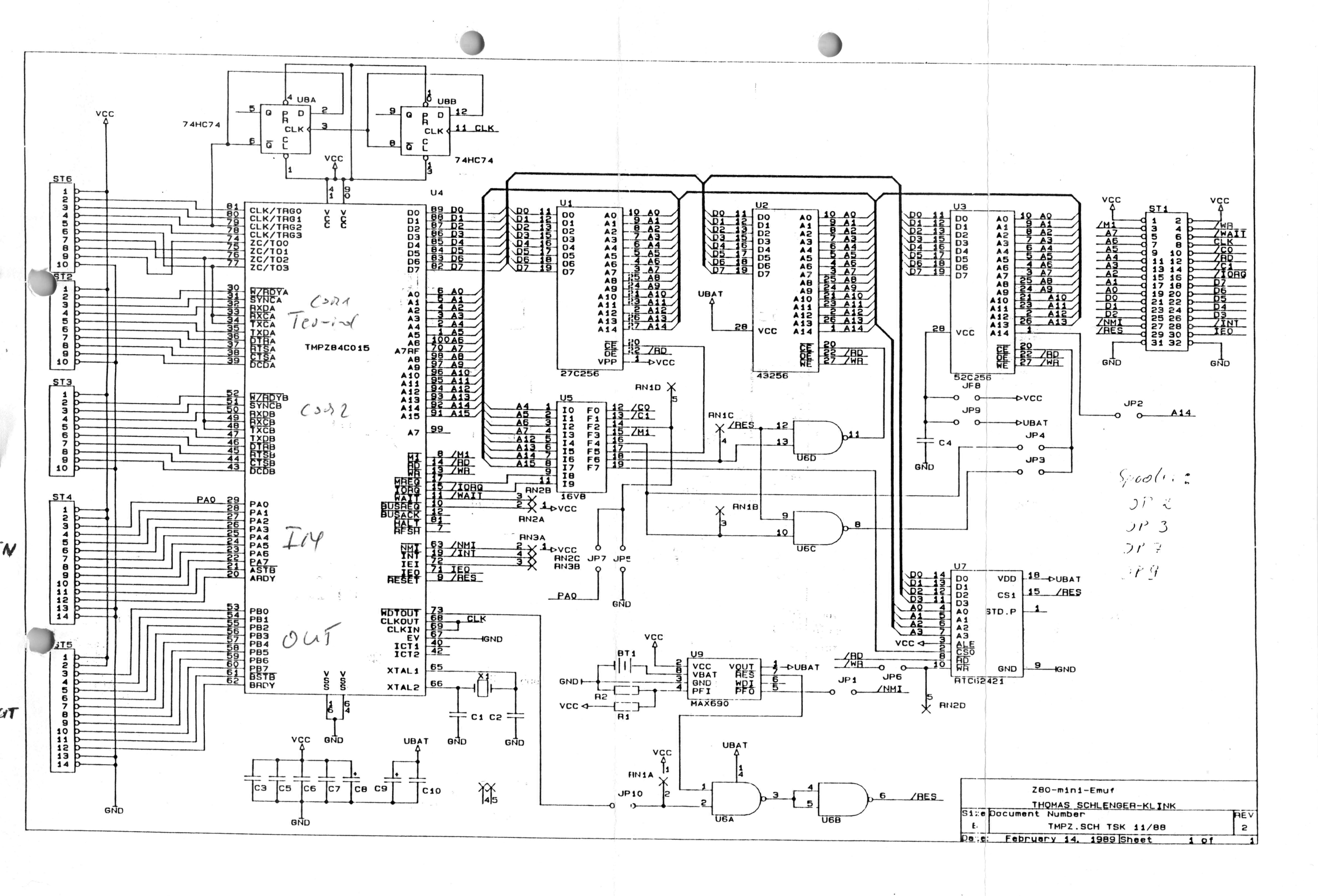
Handbuch

MSR-BASIC

electronic GmbH
Rheingrafenstr. 37 · 6501 Wörrstadt
Tel.: 06732/5029 · Fax: 06732/61496




Z80-mini-Emuf
THOMAS SCHLENGER-KLINK
Document Number
TMPZ.SCH TSK 11/88
REV
2
February 14, 1989
Sheet 1 of 1
TMPZ84C015
27C256
43256
52C256
RTC62421
MAX690
16V8
74HC74

I N H A L T S V E R Z E I C H N I S

Schaubilderverzeichnis

1. VORWORT

Entsprechend den vielfältigen Einsatzmöglichkeiten richtet sich MSR-BASIC an einen weiten Kreis von Anwendern.

Das vorliegende MSR-BASIC-Handbuch stellt hinsichtlich Darstellung und Umfang einen Kompromiß dar.

Diejenigen MSR-BASIC-Befehle, die zum Standard-Sprachumfang von BASIC gehören, werden sehr knapp beschrieben, während die MSR-spezifischen Befehle in der gebotenen Ausführlichkeit behandelt werden.

Die Methodik der Darstellung ist von dem Ziel geprägt, das jeweilige Grundprinzip der einzelnen MSR-BASIC-Befehle knapp und allgemein zu vermitteln, andererseits durch praktische Beispiele Anwendungshinweise für den MSR-Bereich zu liefern.

Aus diesem Grund werden die Befehle nicht in ihrer alphabetischen Reihenfolge beschrieben, sondern im Rahmen funktionell verwandter Befehle, wie z.B.

- Echtzeitbefehle
- Programmflußsteuerbefehle
- Zeichen-Ein-/Ausgabe-Befehle

Das Handbuch kann natürlich kein Ersatz für eine Schulung sein.

Bei entsprechendem Bedarf bitten wir Sie, sich an Ihren MSR-BASIC-Händler bzw. an THERMOPLAN zu richten.

MSRBASIC Version 4.1.4

Die neue MSRBASIC-Version für den Z80-MiniEmuf wir <u>mit</u> dem Monitor in einem EPROM ausgeliefert.

Wie bisher ist zwischen zwei Versionen zu unterscheiden:

MSRBASIC/16 -> verwaltetes BASIC-RAM 16 KByte auf U2 und auf U3 wird ein EEPROM der Typen 2864 oder 28256 (hieraus nur 16KByte) unterstützt.
(Achtung: Jumper JP5 schließen !)

MSRBASIC/32 -> verwaltetes BASIC-RAM 32 KByte auf U2

Für den Monitor <u>und</u> das MSRBASIC ist nun der serielle Kanal A (ST2) die Konsolenschnittstelle.

<u>Wechsel zwischen MSRBASIC und Monitor</u>

MSRBASIC -> Monitor mit dem Kommando "sys" im "nof"-Modus

Monitor -> MSRBASIC mit dem Kommando "g0000"

<u>Ablage eines MSRBASIC-Programm's im EEPROM</u>

Mit dem neuen Kommando **eepromit** wird ein MSRBASIC-Programm ins EEPROM transferiert.

<u>Achtung!</u>

Nach der Ausführung des Kommandos ist das Programm im BASIC-RAM gelöscht. Nach einem erneuten PowerUp oder einem **g0000** aus dem Monitor heraus steht das Programm wieder im BASIC-RAM zur Verfügung.
Das BASIC-Programm wird im EEPROM in tokenisierter Form abgelegt, ist also sehr kompakt. Hier gilt für ein EEPROM des Typ's 2864 (8 KByte):Ist das Programm im BASIC- RAM entwickelbar ist, kann es in jedem Falle im EEPROM abgelegt werden (ein Gegenbeispiel ist noch nicht bekannt).

Ablage eines MSRBASIC-Programm's im EPROM

Mit dem Kommando **promit** ohne Optionen wird das komplette MSR-BASIC und, so vorhanden, ein MSRBASIC-Programm über die Konsolenschnittstelle im Intel-Hex-Format ausgegeben. Will man den Sokkel U3 des Z80-MiniEmuf mit einem Eprom anstelle des vorgesehenen EEPROM benutzen um von dort aus ein MSRBASIC-Programm zu laden, so ist das Kommando **promit &H8000** abzusetzen. Die Intel-Hex kodierten Zeichen sind dann an der Konsolenschnittstelle "aufzufangen" (siehe PDV-TERM) und in einer Datei abzulegen. Diese Datei muß dann mit einem Intel-Hex -fähigen EPROMer in ein 2764 (oder 27256) programmiert werden.
In Zweifelsfällen rufen Sie uns an.

Zeileneditor

Der Zeileneditor wurde überarbeitet sowie ein **auto**-Kommando eingefügt.
Das **auto**-Kommando nummeriert die Zeilen automatisch und besitzt folgende Syntax:

```
auto [ Startzeilennummer [, Schrittweite]] CR
```

Werden keine Optionen angegeben, so benutzt **auto** als Voreinstellung

Startnummer = 10 und Schrittweite = 10

Der Zeileneditor befindet sich grundsätzlich im Einfügemodus.
Folgende Funktionen stehen zur Verfügung:

Cursor	nach links	-> CTRL-A
	nach rechts	-> CTRL-F
	Anfang/Ende einer Zeile	-> CTRL-B
Löschen	Zeichen links	-> DEL, BS
	unter Cursor	-> CTRL-G
	ganze Zeile	-> CTRL-X
Eingabe	beenden	-> CR
	abbrechen	-> CTRL_C

Peek/Pooke-Funktionen

Von den Anwendern wurde immer nach diesen Funktionen gefragt. Sie stellen eine nicht zu unterschätzende Gefahr für ein Programmiersystem dar, das in Prozeßanwendungen sicher funktionieren soll. Da aber die Entmündigung der Anwender nicht unser Ziel ist, wurden diese Funktionen implementiert. Sie sollten aber mit Vorsicht benutzt werden. Nun zur Syntax:

```
a$     = PEEK(addr) AS 2H    ; [ 0 <= addr <= 65535 ]
inhalt = PEEK(addr)

POKE( addr, wert )           ; [ 0 <= addr <= 65535 ]
                             ; [ 0 <= wert <= 255   ]
```

Eingabe/Augabe/Zuweisung von Sedezimalzahlen

Sedezimal(Hex)zahlen können nun wie folgt behandelt werden:

```
Eingabe -> ... AS nH         ; n ( sinnhafte) Anzahl
Ausgabe wie Eingabe

Zuweisung Präfixnotation von &H , Beispiel:

ADDR = &H1000;
```

Arithmetik

Die Arithmetik wurde gänzlich überarbeitet, was zu einer deutlichen Geschwindigkeitssteigerung führte. Die zeitliche Konsequenz kann mit einer Halbierung der Laufzeit bei einem normalen Programm angegeben werden.

Änderungen der Befehlskurzformen

Folgende Befehlskurzformen wurden aus Konfliktgründen geändert:

```
P (PRINT)    -> ?
G (GOTO )    -> GO
I (INPUT)    -> IN
N (NEXT )    -> NX
W (WAIT )    -> WT
R (REM  )    -> REM
```

Weiterhin besitzen alle 3-Buchstaben-Befehle keine Abkürzungen mehr.

NOFREEZE ist jetzt nur noch in der Kurzform NOF verwendbar.

Änderungen der Meldungen

```
'READY                  ' --> '>                                '
'ERROR                  ' --> 'Fehler                           '
'syntax- or value-error' --> 'Fehlerhafte Syntax oder Zuweisung'
'INPUT ERROR, TRY AGAIN' --> 'INPUT Fehler                     '
'MEM FULL               ' --> 'Speicher voll                    '
'BYTE FREE              ' --> 'freie Bytes                      '
```

Änderungen von Funktionsnamen

```
'UPPER_SCREEN           ' --> 'WINDOW'
'TIME_FACTOR            ' --> 'TIMFAC'
```

Weitere Änderungen

'ERRVAR' ist eine Systemvariable, die die letzte Fehlernummer enthält

'ONERROR' UP Definition die bei Fehler ausgeführt wird

```
'USR                    ' --> wird später syntaktisch geändert
'CALL                   ' --> wird später syntaktisch geändert
```

Bei LIST oder SAVE werden hinter Token / Variablen kein zusätzliches Blank mehr ausgegeben, wenn ein ")", "," oder ";" folgt.

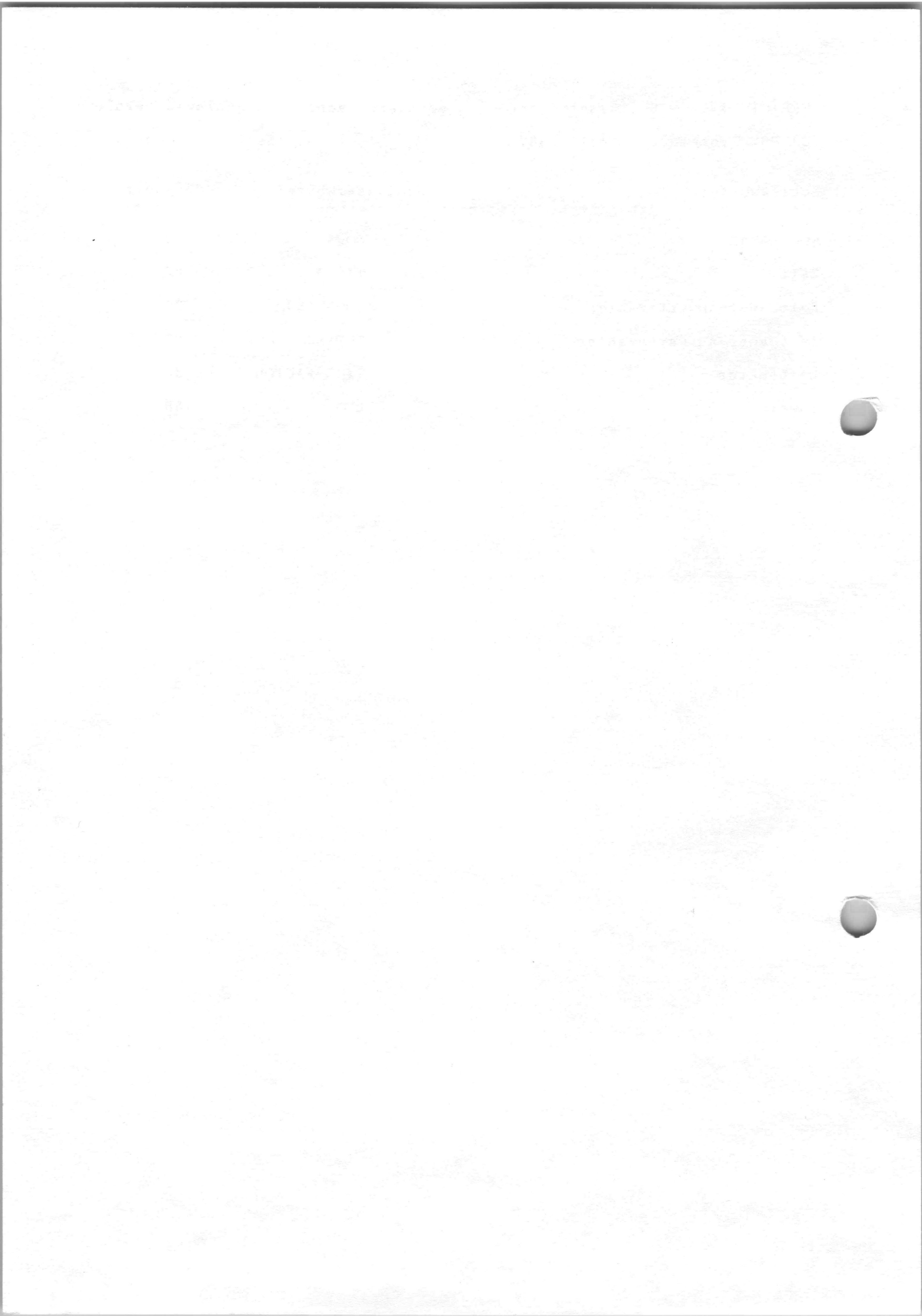
Bei der formattierten Ausgabe werden keine Formatfüller "*" ausgegeben, notfalls werden mehr Stellen ausgegeben oder es wird das xEy Format benutzt.

Neue Funktionen

ASC() Neue Funktion. Gibt die ASCII-Codenummer des ersten Zeichen des Aufrufparameters als Floatzahl zurück. z.B. ASC("ABC") gibt den Wert 65.0 zurück.

SGN (Xe) Neue Schaltreglerfunktion. Liefert -1 oder 1 abhängig vom Vorzeichen des Parameters Xe.

SAT (F, Fmax, Fmin)
Neue Begrenzerfunktion. Liefert Intervallgrenzen, wenn F diese Grenzen über- oder unterschreitet, ansonsten F.



Nachdem in der letzten Ausgabe von MC die Hardware des "endgültigen Z80-EMUF" beschrieben wurde, beschreibt dieser Artikel die bereitgestellte Software - einen pfiffigen Basic-Interpreter mit dem einprägsamen Namen MSRBASIC.

Nicht schon wieder einen BASIC-Interpreter! Sterben diese prähistorischen Gebilde denn nie aus? So, oder ähnlich klingt es mir in den Ohren, während ich an meinem Schreibtisch sitze und darüber nachdenke, wie ich Ihnen den Mund wässrig machen kann. Zugegeben: Als Zeitgenosse der MFLOP's erscheint MSRBASIC ein Anachronismus schlechthin zu sein; aber schauen wir uns das "Urvieh" doch einmal näher an.

Geschichtliches

MSRBASIC wurde am Lehrstuhl für Steuerungs- und Regelungstechnik der Technischen Universität München entwickelt.
Zielintention der Entwicklung war die Schaffung eines Programmiersystems für eine breite Schicht von Anwendern aus den Bereichen Steuerungs-/Regelungstechnik, Verfahrenstechnik, Chemie, Physik, Medizin u.a. . Es sollte also ein Werkzeug konstruiert werden, mit dessen Hilfe man in den o.g. Anwendungsbereichen (programmtechnische) Lösungen von Problemstellungen erarbeiten konnte. Die Entstehung fachfremder (programmtechnischer) Problematiken sollten vermieden werden, damit bei der Benutzung des zu schaffenden Programmiersystems die Konzentration bei der Bearbeitung der fachspezifischen Problemstellung verbleiben konnte.

Werkzeug MSRBASIC in einer Kurzübersicht

Die o.g. Anforderungen sind hoch. Wie stellt sich das Werkzeug MSRBASIC seinem Benutzer nun dar ? Hier eine kurze Übersicht:

- Sprachbasis des MSRBASIC's ist MBASIC von Microsoft
- zusätzliche MSRBASIC Sprachkonstrukte:

  a) TASK   Definition und Planung von Programmteilstücken, die zeitzyklisch bearbeitet werden, Anzahl 5

b) SEQUENZ Definition und Planung von Programmteilstücken, in denen Weiterschaltbedingungen formuliert werden können. Mit diesem Sprachkonstrukt lassen sich häufig benötigte Ablaufsteuerungen realisieren, Anzahl 25

c) DIN(KN) DigitalerINput(KanalNummer)
DOUT(KN) DigitalerOUTput(KanalNummer)
ADC(KN) AnalogDigitalConverter(KanalNummer)
DAC(KN) DigitalAnalogConverter(KanalNummer)
Diese Sprachkonstrukte ermöglichen einen hardwareunabhängigen Zugriff auf die Prozeßperipherie. Die Benutzung dieser Konstrukte setzt wohl den Anschluß einer vorliegenden Hardware an die Sprachkonstrukte im Rahmen einer Implementierung voraus, fordert dabei aber Spezialisten und nicht den Benutzer.

d) TUP(N) TimerUP(Nummer)
TDOWN(N) TimerDOWN(Nummer)
Zeitdauer-Aufwärts/Abwärts-Zähler die als eindimensionale Felder vereinbart werden können, gezähltes Zeitquantum ist 10 ms. Die Anzahl der Zähler ist nur vom Speicherplatz beschränkt.

e) PI-
PID- Reglerfunktionen, jeweils in einem Stellungs- oder einem Geschwindigkeitsalgorithmus. Die Reglerfunktionen sind vom Benutzer nur mit den notwendigen Parametern zu versehen.

f) Vektor-
Matrix- Arithmetik, diese fast mit max. Prozessorgeschwindigkeit

g) Status- und Kommandofunktionen für serielle Schnittstellen

- zusätzliche Fähigkeiten

a) Autoload- und Autostartfunktion
b) EPROM-fähiger Zwischencode

Nun ist die Kurzübersicht wohl eher zu einer Langübersicht geraten. Sollten Sie aus der Kurzübersicht den Verdacht gewonnen haben, daß MSRBASIC "echtzeitfähig" ist, liegen Sie richtig.
Echtzeit- und multitaskingfähiges MSRBASIC

Programmiersysteme, die in der Lage sind, auf ein Prozeßereignis in einer vorhersagbaren Zeit zu reagieren (Ereignierkennung und Bereitstellung der für diese Situation erforderlichen Daten) , nennt man "echtzeitfähig". Dabei ist die tatsächliche Dauer der Reaktionszeit nebensächlich, wichtig ist allein, daß in einer vorhersagbaren Zeit reagiert werden kann. Die tatsächliche Dauer der Reaktionszeit sagt nur etwas über die Leistungsfähigkeit eines "echtzeitfähigen" Programmiersystemes aus (siehe DIN 44 300).

Echtzeitfähigkeit ist gut - Multitaskingfähigkeit aber etwas anderes. Zum Multitasking kam man vor dem Hintergrund der Erkenntnis, daß die CPU eines Rechners selten richtig ausgelastet ist (z.B. Kommunikation mit langsamen Peripheriegeräten u.a.). Wäre man in der Lage, der nach " sinnhaften" Tätigkeiten schmachtenden CPU eine andere Aufgabe (task) zur Bearbeitung zu übergeben, könnte die sich, statt Däumchen zu drehen, schon einmal damit beschäftigen. Begnügt man sich nicht mit dem: "Wäre man in der Lage ..." muß man sich eine Verwaltungsinstanz konstruieren, die die Aktivitäten der CPU in geordnete Bahnen lenkt. Derartige Verwaltungsinstanzen nennt man in Multitaskingsystemen Prozeßumschalter oder auch Dispatcher. An ein echtzeit- und multitaskingfähiges Programmiersystem sind somit folgende Forderungen zu stellen:

a) Es müssen Sprachmittel zur Verfügung stehen, die die Formulierung von mehreren Aufgaben (weiterhin Task's) gestatten.
b) Es müssen Sprachmittel zur Verfügung stehen, die die zeitliche oder ereignisorientierte Ein- bzw. Ausplanung der Task's gestatten.
c) Es muß eine Instanz vorhanden sein, die die formulierten Ein- bzw. Ausplanungen zur Ausführung bringt und nach Möglichkeit Mittel zur Beseitigung programmtechnischer Fehler bereitstellt.

MSRBASIC-Sprachmittel zu Punkt a)

MSRBASIC bietet die Vereinbarung von Task's und die Vereinbarung von Sequenzen an. Task's und Sequenzen werden wie BASIC-Unterprogramme formuliert mit dem Unterschied, daß sie der Planungsinstanz vor Benutzung bekanntgegeben werden müssen.

Beispiel:

```
10   REM Beispiel der Task und Sequenzvereinbarung
20   REM Initialisierung verwendeter Variablen
30   Tp = 1.0 : INP$ = "N" : SER-1 = 0
40   REM Tp soll die PlanungsZeit-Variable sein, mit der
50   REM Bedingung 0.01s <= Tp <= 2.54s
60   REM
70   DEFINE TASK 1, Tp , 2000
80   DEFINE SEQUENZ 1, 3000
90   UPPER-SCREEN = 10
100  REM
110  ACTIVATE TASK 1, SEQUENZ 1
120  REM
130  STOP
.
.
.
2000 REM Hier ist die Aufgabe der TASK 1 formuliert
2010 REM
2020 PRINT ">>> Task 1: -> Testausgabe <- "
2030 REM
2040 RETURN
.
.
.
3000 REM Hier ist die Aufgabe der SEQUENZ 1 formuliert
3010 REM
3020 WAIT FOR INP$ ="J"
3030 SUSPEND TASK 1
3040 PRINT
3050 PRINT ">> SEQUENZ 1: TASK 1 wurde suspendiert";
3060 PRINT
3070 REM
3080 RETURN
.
.
.
4000 END
```

In der Zeile 70, des vorstehenden Programmbeispiels, wird die TASK-1 der MSRBASIC-Planungsinstanz "REALTIME-MONITOR" bekanntgegeben durch das Schlüsselwort DEFINE, gefolgt von der Angabe der TASK-Nummer nach dem Schlüsselwort TASK , des Aufrufintervalls und der Zeilen-Nummer, ab der der TASK-Programmkörper formuliert ist bzw. wird. Das Aufrufintervall wird als Real-Zahl notiert mit den in Zeile 50 angegebenen Grenzen. In ähnlicher Weise wird nun eine Sequenz vereinbart (Zeile 80) wobei die Angabe des Aufrufintervalls entfällt.
In Zeile 90 wird der MSRBASIC-Systemvariable UPPER-SCREEN der Wert 10 zugewiesen und damit gleichzeitig die Bildschirmfensterverwaltung aktiviert, die nun dafür sorgt, daß alle programmtechnischen Zeichenausgaben auf dem Konsolen-Kanal in ein oberes Fenster geschrieben werden (hier 10 Zeilen groß) und alle Dialoge mit dem Interpreter über das untere Fenster abgewickelt werden (hier 24-10=14 Zeilen). Voraussetzung für diesen Zwei-Fensterbetrieb ist ein TVI925-Terminal bzw. ein diesen Terminaltyp emulierendes Terminalprogramm.
Weiterhin wird durch die Anweisung in Zeile 110 dem REALTIME-MONITOR durch das Schlüsselwort ACTIVATE mitgeteilt, daß er nun für die Ausführung der deklarierten TASK-1 und der SEQUENZ-1 zu sorgen hat.
Da für den MSRBASIC-Interpreter direkt nichts mehr zu tun ist, wird er durch die Anweisung STOP in Zeile 130 an weiteren Aktivitäten gehindert. Diese Anweisung ist wichtig, weil der Interpreter sonst nicht davon abgehalten werden könnte, den ab Zeile 2000 formulierten TASK-Programmkörper zu exekutieren. Wohin soll er aber zurückkehren, wenn er in Zeile 2040 auf die Anweisung RETURN trifft ?
Die ab Zeile 2000 bis 2040 formulierte TASK-1 macht nun nichts anderes, als gemäß ihrer Vereinbarung im Zeitabstand von Tp= 1.0 (zeitzyklisch) die in Zeile 2020 formulierte Meldung in das obere Bildschirmfenster zu schreiben.
Gleichzeitig lauert die ab Zeile 3000 bis 3080 formulierte SEQUENZ-1 in der Zeile 3020 darauf, daß die Stringvariable INP$ irgendwann einmal den Großbuchstaben "J" enthält. Der nachfolgende Programmkörper der SEQUENZ-1 wird also erst dann ausgeführt, wenn diese "Weiterschaltbedingung" erfüllt ist. Nun überlasse ich es dem Leser selbst nachzuvollziehen was passiert, wenn die String-

variable INP$ durch eine Zuweisung den Inhalt "J" erhält.
MSRBASIC-Multitasking, Programmtypen und ihre Prioritäten

Bei der Betrachtung des Beispielprogrammes gewinnt man diffus den Eindruck, daß unterschiedliche Ablaufstrategien die Funktion des Gesamtprogrammes bestimmen. Strukturieren wir das:

a) Das Programmstück ab Zeile 10 bis Zeile 130 wird offensichtlich sofort nach einem Programmstart (Kommando RUN) durchlaufen.

b) Das Programmstück ab Zeile 2000 bis Zeile 2040 ist als TASK-1 vereinbart und wird durch den REALTIME-MONITOR entsprechend seines Aufrufintervalles zeitzyklisch zur Ausführung gebracht.

c) Das Programmstück ab Zeile 3000 bis Zeile 3080 ist als SEQUENZ-1 vereinbart und wird in seinem Laufverhalten vom SEQUENZ-HANDLER, einem spezialisierten Programmteils des REALTIME-MONITORS gesteuert. Immer dann, wenn eine formulierte Weiterschaltbedingung erfüllt ist, wird der nachfolgende Sequenz-Programmkode zur Ausführung freigegeben.

Drei Programmtypen - eine CPU ! Wie soll das Funktionieren ?

Die o.a. Verwaltungseinheit, der REALTIME-MONITOR ist die Rettung. Zuerst verlangt der mal klare Verhältnisse, d.h, der nimmt erst seine Arbeit auf, wenn man das FREEZE-Kommando eingegeben hat. Durch dieses Kommando wird der Speicher entrümpelt, ein Edieren des Programmkodes sowie die Schaffung neuer Datenobjekte verhindert. Nach dem FREEZE-Kommando kann ein Programmierer dem REALTIME-MONITOR nicht mehr unkontrolliert ins Handwerk pfuschen. Dieser Zustand bleibt solange erhalten, bis ein NOFREEZE-Kommando eingegeben wird.
Alsdann teilt der REALTIME-MONITOR die Rechenzeit der CPU in kleine Häppchen, sprich Zeitscheiben, von der Größe 10ms . Eine solche Scheibe nennt er Planungsatom und für den Programmierer folgt daraus, daß jede versuchte Einplanung einer TASK mit einer Zykluszeit < Planungsatom mit einer Fehlermeldung bedacht wird.
Zum Schluß legt der REALTIME-MONITOR noch fest, nach welcher

Priorität er die Rechenzeithäppchen an die verschiedenen Programmtypen (s.o. a,b und c) verteilt und teilt mit:

Zuerst bekommt	der MSRBASIC-Programmtyp TASK entsprechend der impliziten Prioritäten Rechenzeit (TASK-1 besitzt die höchste - TASK-5 die niedrigste Priorität)
dann	der MSRBASIC-Programmtyp SEQUENZ und
letztlich	der MSRBASIC-Standard-Interpreter.

Im Handbuch wird der MSRBASIC-Standard-Interpreter im Echtzeitbetrieb auch Hintergrundprogramm genannt. Er ist aber in der Multitasking-Umgebung nichts anderes als der Prozeß mit der niedrigsten Priorität und lebt nur vom Rechenzeitabfall anderer Prozesse, wobei er als Sonderfall auch der einzige Prozeß sein kann.

In der vorliegenden Implementation des MSRBASIC's verwaltet der REALTIME-MONITOR 5 TASK's und 25 SEQUENZen. An der Stelle mal schnell eine Warnung: Obwohl leistungsfähig ist der "endgültige Z80-EMUF" keine Micro-VAX. Deshalb verbietet sich die Konstruktion von 5 Task's mit einem Aufrufintervall = Planungsatom. Die Taskprogrammlaufzeit und ihr Aufrufintervall sollten auch in einem sinnhaften Zusammenhang stehen. Zur Fehlerbeseitigung in einer zweifelhaften Situation stehen die Fehlersuchkommandos TASKLIST und SEQLIST zur Verfügung. Genug des Multitaskings ! MSRBASIC hat ja auch noch anderes zu bieten.

MSRBASIC und die Prozeßperipherie

MSRBASIC greift auf die Prozeßperipherie über 4 skalare Einlese- und 4 skalare Ausgabefunktionen zu. Skalar, weil sie im Gegensatz zu den vektoriellen Ein-/Ausgabefunktionen nur einen Wert liefern.

Einlesefunktionen	Ausgabefunktionen
ADC(KanalNummer)	DAC(KanalNummer)
DIN(KanalNummer)	DOUT(KanalNummer)
GET(I/O-PortNummer)	PUT(I/O-PortNummer)
CNT(KanalNummer)	CNT(KanalNummer)

Handhabung der Prozeßperipherie mit MSRBASIC-Sprachmittel

Im Interpreter Direkt-Modus liefert bzw. setzt/gibt:

```
PRINT ADC(1) -> normierte Real-Zahl des analogen Eingangskanals 1
PRINT DIN(4) -> den Wert 0 oder 1 des Binärkanals 4
PRINT GET(8) -> den Zustand des CPU-I/O-Ports 8
PRINT CNT(2) -> den Zählerstand eines 16 Bit Zählerkanals 2

DAC(15) = 0.5   den analogen Ausgangskanal 15 auf Full-Scale / 2
DOUT(2) = 1     den Binärkanal 2 auf den log. Wert 1
PUT(12) = 128   an dem CPU-I/O-Port 12 die Zahl 128 aus
CNT(3)  = 200   den Zählerkanal 3 auf den Zählerstand 200
```

Die Anschlüsse dieser MSRBASIC-Sprachmittel an die reale Hardware des Ablaufrechners ist Aufgabe der jeweiligen Implementation. Über die Funktionen NADC, NDAC, NDIN, NDOUT und NCNT kann man sich auf jedem MSRBASIC-System Aufschluß über die Anzahl der Prozeßperipherie-Kanäle verschaffen.

Die Leistungsfähigkeit der MSRBASIC-Prozeßzugriffsfunktionen läßt sich an einem kleinem Beispiel demonstrieren. Es soll eine Pumpe in Abhängigkeit der Schalter S1, S2, S3 und S4 geschaltet werden. Der funktionelle Zusammenghang sei durch folgende Boolsche Gleichung beschrieben:

Pumpe = S1 * (S2 + /S3 + S4)

Abbildung in einem MSRBASIC-Programm:

```
10  REM Pumpe wird durch den digitalen Ausgangskanal 5 geschaltet
20  REM
30  Pumpe = 5
40  REM
50  REM Die Schalterstellungen werden über digit. Eingangskanäle
60  REM 1...4 eingelesen
70  REM
80  S1 = 1 : S2 = 2 : S3 = 3 : S4 = 4
90  REM
```

```
100 DOUT(Pumpe) = DIN(S1) * (DIN(S2) + NOT(DIN(S3)) + DIN(S4))
```

Die Zeilen 10 bis 90 des vorstehenden Beispiels bereiten nur die Umsetzung der Boolschen Gleichung in Zeile 100 vor und steigern die Lesbarkeit dieser Zeile. Die Nähe der programmtechnischen Realisation zur Problemstellung ist offensichtlich.

Zum Abschluß der Betrachtung MSRBASIC und die Prozeßperipherie noch ein Blick auf die vektoriellen Einlese- und Ausgabefunktionen. Hierfür stellt MSRBASIC folgende Funktionen zur Verfügung:

MADC( ), MDAC( ), MDIN( ) und MDOUT( )

Argumente dieser vektoriellen E/A-Funktionen sind vereinbarte Vektoren, die Meß- oder Stellortvektoren genannt werden. Kleines Beispiel mal ohne REMarks:

```
10 N=10
20 DIM IN(N), MO(N), OUT(N), SO(N)
30 VEC  IN = MADC(MO) ,  MDAC(SO) = OUT
.
.
1000 END
```

Vektoren werden wie eindimensionale Felder vereinbart. Ihren vektoriellen Charakter erhalten sie erst im Zusammenhang mit den vektoriellen Operationen. In Zeile 30 des Programmbeispiels werden durch das Schlüsselwort VEC und den Funktionen MADC( ) bzw. MDAC( ) folgende Einzel-Operationen ausgelöst:

```
IN(1)=ADC(1), IN(2)=ADC(2), ... IN(N-1)=ADC(N-1), IN(N)=ADC(N)
                            und
DAC(1)=OUT(1), DAC(2)=OUT(2),... DAC(N-1)=OUT(N-1),DAC(n)=OUT(N)
```

Ist doch ganz nett was man durch eine einzige MSRBASIC-Anweisung alles auslösen kann. Nun mal schnell zu den parametrierbaren Reglerfunktionsblöcken.

MSRBASIC und PI- bzw. PID-Regler

Wie schon fast erwartet erledigt MSRBASIC auch diese Kleinigkeit

mit fast einer Zeile. Die Reglerfunktionen PI( ), PID( ), GPI( ) und GPID( ) werden mit der momentanen Regeldifferenz E als Eingangsgröße sowie mit den Reglerparameter Kp, Ki, (Kd), Yo, Yu und Yd aufgerufen und reichen eine Stellgröße Y heraus. Für die Zustandsgrößen sind max. noch drei Variable bereitzustellen, was aber in einem BASIC-Interpreter kein Problem darstellt. Ein schnell programmierter Regler könnte wie folgt aussehen:

```
.
.
100 E = SOLL - ADC(IST)
110 DAC(Stell) = PID(Y,E,E1,E2,Kp,Kd,Ki,Yo,Yu,Yd,AK)
120 GOTO 100
.
.
```

Die detailierte Beschreibung der Reglerfunktionsblöcke entnehmen Sie bitte dem MSRBASIC-Handbuch. Auch die Reglerfunktionen können vektorisiert verwendet werden; sollte man auch tun wenn man mehrere Regler benötigt (erhöht die Ausführungsgeschwindigkeit).

MSRBASIC besitzt noch eine ganze Menge beschreibenswerter Fähigkeiten, (so die Vektor-/Matrix-Arithmetik, die Ansprache der seriellen Datenkanäle über log. Nummer - PRINT ON(3) -, Status und Kommandofunktionen für die seriellen Schnittstellen, je nach Implementation auch Kommunikationsfunktionen zum Vernetzen von MSRBASIC-Rechnern u.a.m.) die aber an dieser Stelle nicht mehr beschrieben werden sollen. Nachfolgend soll nun das Verhältnis des "endgültigen Z80-EMUF" zum MSRBASIC beschrieben werden.

Der "endgültige Z80-EMUF" und MSRBASIC

Die drei Speichersockel (U1, U2 und U3) des "endgültigen Z80-EMUF's" werden unter MSRBASIC wie folgt benutzt:

Speicherlayout

Standardlayout

U1	0000H ... 07FFH	-> EPROM	27C256	
U2	8000H ... FFFFH	-> SRAM gepuffert	62256	JP5 offen

Speicherlayout 1 - EPROM auf U3

U1	0000H ... 07FFH	-> EPROM	27C256	
U2	8000H ... BFFFH	-> SRAM gepuffert	62256	JP5 geschlossen
U3	C000H ... FFFFH	-> EPROM	27C256	

Speicherlayout 2 - EEPROM auf U3

U1	0000H ... 07FFH	-> EPROM	27C256	
U2	8000H ... BFFFH	-> SRAM gepuffert	62256	JP5 geschlossen
U3	C000H ... DFFFH	-> EEPROM	2864	
	... FFFFH	-> EEPROM	28256	

Im Standardspeicherlayout residieren das Monitorprogramm (beschrieben in MC ****) und der MSRBASIC-Interpreter im EPROM, einem 27C256 auf Speichersockel U1. Als Arbeitsspeicher stehen den beiden Programmen ein SRAM 62256 im Speichersockel U2 akkugepuffert zur Verfügung. Aus dem MSRBASIC gelangt man mit dem Kommando SYST in den Monitor, aus dem Monitor mit dem Kommando B zurück ins MSRBASIC. Die Datenbereiche beider Programme sind getrennt, sodaß man unproblematisch hin und her springen kann. Benutzt man das Standardspeicherlayout stehen rund 29 KByte BA-SIC-RAM-Bereich zur Verfügung, der zudem noch akkugepuffert ist.

Das Speicherlayout 1 - EPROM auf U3

Im Unterschied zum Standardspeicherlayout stehen hier nur die unteren 16 KByte des 62256 (U2) zur Verfügung. Der Jumper JP5 ist zu schließen, damit die Adreßdekodierung die obersten 16 KByte (C000H...FFFFH) dem Speichersockel U3 zuordnet. Dann sind für den EPROM-Einsatz auf Sockel U3 noch folgende Modifikationen vorzunehmen: JP2 öfnen, Signalleitung /WR von PIN27-U3 entfernen,

PIN27-U3 (A14) mit Masse und PIN1-U3 mit +5V verbinden. Sinn und Zweck des Speicherlayouts 1 siehe MSRBASIC Autostart. Bei diesem Speicherlayout stehen rund 12,5 KByte BASIC-RAM-Bereich zur Verfügung.

Das Speicherlayout 2 - EEPROM auf U3

Speicherlayout 2 entspricht dem Speicherlayout 1 mit dem Unterschied, daß eine Hardwaremodifikation nicht erforderlich ist. Jumper JP5 ist zu schließen (s. Speicherlayout 1). MSRBASIC unterstützt in diesem Speicherlayout den Einsatz von EEPROM's der Typen 2864 und 28256 (bei den 28256 werden nur 16 KByte benutzt).

MSRBASIC Autostart

MSRBASIC ist autostartfähig und benutzt in dieser Implementation folgende Mechanismen:

a) P - Autostartmodus : Bei einem Kaltstart (Versorgungsspannung ein) überprüft MSRBASIC, ob in seinem BASIC-RAM-Speicher ein BASIC-Programm steht. Ist das der Fall, schaltet MSRBASIC den FREEZE-Modus ein und exekutiert das Programm. Da beim "endgültigen Z80-EMUF" der RAM-Sockel U2 gepuffert ist, bietet der P-Autostartmodus eine einfache Möglichkeit einen Autostartbetrieb zu realisieren. Für diesen Einsatzfall ist das Standardspeicherlayout vorgesehen, das einen BASIC-RAM-Bereich von 29 KByte zur Verfügung stellt.

b) L - Autostartmodus : Dieser load-and-go-Modus wird dann aktiv, wenn kein BASIC-Programm im BASIC-RAM gefunden wird. Über einen vereinbarten seriellen Kanal erwartet MSRBASIC die Eingabe eines BASIC-Programmes. Hier sind nun zwei Möglichkeiten implementiert:

L - EPROM : Sie benutzen das Speicherlayout 1, schreiben mit einem ASCII-Editor ein MSRBASIC-Programm, wobei Sie nur darauf achten, daß die Einer-Stelle der Programmzeilennummer in Spalte 6 steht (z.B. 10 PRINT TEST als: ----10 PRINT TEST, wobei die underline-Zeichen in diesem Beispiel für blank-Zeichen stehen). Am Ende des so erstellten Programmes fügen

Sie ein ^T (14H) ein und schreiben diese Datei in ein EPROM vom Typ 27C256. Dieses EPROM stecken Sie in den Sockel U3, schalten die Versorgungsspannung des "endgültigen Z80-EMUF's" ein und schon wird das Programm exekutiert.

L - EEPROM : Sie benutzen das Speicherlayout 2, erstellen mit dem MSRBASIC-Editor ein Programm und geben das Kommando: SAVE ON(2) ein. Ein evtl. im EEPROM stehendes Programm wird Ihnen angezeigt. Geben Sie nun das Zeichen ^A (01H) ein, wird das im BASIC-RAM stehende Programm im EEPROM abgespeichert. Soll ein Übertschreiben des EEPROM-Programmes verhindert werden, brechen Sie mit ^C (03H) das SAVE ON(2)-Kommando ab. Beim Einsatz eines EEPROM's vom Typ 2864 können Sie so max. 7 KByte lange Programme im EEPROM speichern, verwenden Sie ein 28256 läßt sich der ganze verwaltete BASIC-RAM-Bereich im EEPROM sichern.

Für beide L - Autostartmodi gilt: Ist ein BASIC-Programm im BASIC-RAM-Bereich, dann wird das nach POWER-ON exekutiert. Befindet sich kein Programm im BASIC-RAM-Bereich, wird versucht aus dem E(E)PROM ein Programm in den BASIC-RAM-Bereich zu kopieren, das dann daraus exekutiert wird. Findet MSRBASIC weder im BASIC-RAM-Bereich noch im E(E)PROM-Bereich ein Programm, startet der Interpreter im Kaltstart-Modus.

übersicht MSRBASIC-Implementation "endgültiger Z80-EMUF"

```
Serielle-Kanäle : SIO-A   LU=0  Konsolenkanal, TVI 925
                : SIO-B   LU=1
                : Parameter 9600,8,2,NO
                : beide Kanäle interruptgesteuert
                :
                : E(E)PROM LU=2  SAVE ON(2)
                :                LOAD ON(2)
                :                LIST on(2)
                : Programmspeicherung
                :
                : EEPROM   LU=3  PRINT ON(3)
                :                INPUT ON(3)
                : Datenspeicherung max. 1 KByte
```

```
                    :
    Prozeß-Kanäle   : PIO-A  DIN(1...8)
                    : PIO-B  DOUT(1...8)
                    :
    Hardware-Uhr    : PRINT time$, date$ , dow$
                    : now$ = time$ : heute$ = date$ : Tag$ = dow$
                    : RTC lesen
                    :
                    : time$ = "12:12:12" : date$ = "31.03.1989"
                    : dow$ = "FRI"
                    : RTC schreiben
                    :
                    : PUT(15) = 15
                    : PUT(15) = 12 -> Test-Modus der Uhr
                    :
                    : PUT(15) = 15
                    : PUT(15) = 4  -> Normalbetrieb der Uhr

Ser. Schnittstellen : Vorausetzungen
                      IF232 Brücke PIN 1 mit PIN 6
                            oder Terminal muß DCD setzen
                      Beide Kanäle Hardwarehandshake RTS/CTS
                      Kanal B zusätzlich XON/XOFF
```

Abschließend möchte ich nicht versäumen Ihnen mitzuteilen, daß MSRBASIC nicht nur auf dem "endgültigen Z80-EMUF" läuft sonder auch auf CPM-Rechnern und auf DOS-Rechnern, daß es eine MSRBASIC-SHELL gibt (Terminal, Editor und up-/download) und es reichlich Prozeßperipherie für den "endgültigen Z80-EMUF" geben wird, die einfach über MSRBASIC gehandhabt werden kann.

Autor: Hans Metzmacher
Joh.-Seb.-Bach-Str. 13
4920 Lemgo
Tel. 05261-12754

Anpassung der MSRBASIC-Konfiguration:

```
Verwalteter RAM-Bereich  16KByte -> Adresse 100BH Inhalt BFBFH
                         32KByte ->         100BH        FFBFH

                         Achtung: LSB vor MSB
```

CTC-Kanäle 2 und 3 Bauderateeinstellung

```
CTC2 SIO-A Adresse 117AH Inhalt 04H -> 9600 bps
                                08H -> 4800 bps
                                10H -> 2400 bps
                                20H -> 1200 bps... SIO /16

CTC3 SIO-B Adresse 1182H Inhalt 04H -> 9600 bps
                                08H -> 4800 bps
                                10H -> 2400 bps
                                20H -> 1200 bps... SIO /16
```

SIO-Initialisierung   Ab Adresse 1A43H befindet sich folgende
                           Tabelle im EPROM:

```
;-------------------------------;
; SIO-INITAB bei Interruptbetrieb
;-------------------------------;
SIOTAB: DB X'38'       *       ;RETI
                               ;
        DB X'18'       *       ;CHANNELRESET
        DB X'04'       *       ;WRITE REG 4
        DB X'4C'               ;4CH -> /16, 2 stop, no par.
        DB X'05'       *       ;WRITE REG 5
CMD5:   DB X'EA'               ;EAH -> dtr, transmit 8 bit,
                                      tx enable, rts
        DB X'03'       *       ;WRITE REG 3
        DB X'E1'               ;E1H -> receive 8 bit, auto,
                                      rx enable
        DB X'01'       *       ;WRITE REG 1
        DB X'18'       *       ;INT ON EVERY Rx
```

```
        DB X'20'        *       ;INT ENABLE
                                ;
;-------------------------------;
```

Die mit * gekennzeichneten Stellen dürfen unter keinen Umständen geändert werden.

Lemgo, im April 1989

2. ALLGEMEINES

2.1 SINN UND ZWECK VON MSR-BASIC

Der MSR-BASIC-Interpreter ist konzipiert für die flexible Implementierung von Meß-, Steuer- und Regelalgorithmen auf Mikrorechnern der Z80- oder der 8086-Familie (stand-alone-Systeme, CP/M-Personal-Computer u.a.).

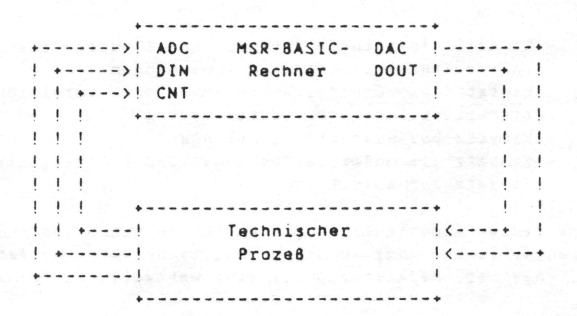


Schaubild 2-1: Universelle MSR-BASIC-Anwendungskonfiguration

Er bietet daher für dieses Einsatzgebiet neben dem üblichen algorithmisch logischen Kern einer höheren Programmiersprache spezielle Sprachmittel an, die die Formulierung von

- nebenläufigen Programmen (Multitasking),
- hardwareunabhängigen Prozeßzugriffen,
- Zeitgebern,
- PI(D)-Reglerfunktionen,
- Matrix-Vektor-Operationen
- Kommunikationsfunktionen

erlauben.

Neben diesen sprachlichen Fähigkeiten, die MSRBASIC als echte Realzeitsprache ausweisen, verfügt der MSRBASIC-Interpreter (im Sinne eines Programmiersystems) über eine Reihe von Eigenschaften, die nicht nur die Erstellung, sondern auch Fehlersuche, Wartung und (Re-)Dokumentation von MSR-Anwenderprogrammen unterstützen:

- Integrität von Ziel- und Entwicklungssystem
- integrierter serieller Hostrechneranschluß (Programm laden, Daten versenden und entgegennehmen, Betrieb als intelligentes Terminal, LAN-Betrieb)
- strukturorientiertes Programm-Ausdrucken
- Befehlsanzeige vor Ausführung zum Programmtest (TRACE-Mode)
- Anzeige des momentanen Programmzeigerstandes bei SEQuenzen (Aufspüren von Deadlocksituationen)
- Fehlermeldung mit Quellzeilen-Auslisten
- E/A-Simulator-Betriebsart

Weitere für Realzeit-Mikrorechner notwendige Eigenschaften sind

- Rechengeschwindigkeitssteigerung durch Einsatz eines Arithmetik-Coprozessors (8087, 8231/9511)
- EPROM-fähiger Zwischencode
- Autoload- und -startfunktion

Für den MSRBASIC-Interpreter ergibt sich damit ein breites Einsatzspektrum:

- Einsatz in autonomen Regel- und Steuergeräten, insbesondere bei verfahrenstechnischen Anwendungen
- Einsatz als Controller zusammen mit intelligenten Peripheriegeräten zur Experimentsteuerung
- Einsatz bei Maschinensteuerungen
- Einsatz als Meßwerterfassungs- und Protokollierungssystem
- Einsatz zur Ausbildung

Die Summe dieser Eigenschaften machen ihn zu einem Instrument, das für viele Anwender (Steuerungs- und Regelungstechniker, Verfahrenstechniker, Chemiker, Physiker, Mediziner u.a.) eine wertvolle Hilfe darstellt.

Das vorliegende Handbuch will in konzentrierter, aber hinreichend ausführlicher Form den Anwender bei der Handhabung des MSRBASIC-Interpreters unterstützen.

Es setzt eine gewisse Vertrautheit mit der Grundsprache BASIC voraus und legt daher das Hauptgewicht auf die Behandlung der echtzeitspezifischen Fähigkeiten von MSRBASIC.

2.2 ANWEISUNGEN, DATENTYPEN und SONSTIGES

In diesem Kapitel werden zunächst - im Sinne einer Übersicht - in alphabetischer Ordnung kurz einige allgemeine MSRBASIC-Eigenschaften behandelt, bevor im zweiten Teil ausführlich das MSR-BASIC-Echtzeitkonzept erläutert wird.

2.2.1 ANWEISUNGEN

Ein MSR-BASIC-Programm besteht aus einer eindeutigen Folge von Anweisungen (Statements). Jede Programm-Anweisung beginnt mit einer Zeilennummer (1...32766), die die Lage der Zeile innerhalb des Programms festlegt.

Einige Anweisungen können auch direkt ("direct mode"), d.h. quasi als Kommando eingegeben werden.

Diese Eigenschaft ist insbesondere beim Echtzeitbetrieb von großer Bedeutung, da während der Programmausführung auf alle Variablen mit Hilfe von PRINT- und (LET-)Anweisungen zugegriffen werden kann.

2.2.2 BUCHSTABEN

MSR-BASIC läßt die Verwendung von Klein- und Großbuchstaben zu. Bei Schlüsselwörtern (Kommandos, Anweisungen, Funktionen, Trennwörtern und Systemvariablen) werden Kleinbuchstaben bei der Eingabe in Großbuchstaben umgewandelt. D.h. Eingaben wie "THEN", "then" oder "Then" werden intern als "THEN" behandelt.

Dagegen werden Kleinbuchstaben in Variablennamen NICHT wie Großbuchstaben behandelt, d.h.

```
            ZEIT$
und                         sind zwei unterschiedliche Variable!
            Zeit$
```

2.2.3 DATENTYPEN

MSRBASIC kennt die Datentypen REAL und STRING. Integer-Zahlen werden intern als Gleitkommagrößen behandelt. Der Datentyp ergibt sich implizit aus dem Programm.

2.2.4 DATENSTRUKTUREN

Als Datenstrukturen sind ein- oder zweidimensionale Felder (s.d.) vom Typ REAL oder STRING möglich.

2.2.5 EDITOR

Bei der Engabe eines Kommandos, einer Programmzeile oder eines Wertes (Zahl oder Text) ist ein Zeileneditor wirksam, d.h. daß innerhalb der laufenden Eingabezeile der Kursor bewegt, Zeichen eingefügt und gelöscht werden können.

Einzelheiten siehe Kapitel "Editor":

2.2.6 FELDER

Es können ein- und zweidimensionale Felder für die Datentypen REAL und STRING benutzt werden. Der Index beginnt grundsätzlich bei 0. Sofern der verfügbare Anwenderspeicher es zuläßt, können numerische Vektoren bis zur Dimension 8191 und Matrizen bis zur Ordnung 126x126 vereinbart werden. Stringfelder können maximal 255 Elemente aufweisen.

2.2.7 FUNKTIONEN

Neben den BASIC-Standardfunktionen wie SIN, SQR bzw. CHR$, MID$ usw. werden folgende Typen von anwendungsbezogenen Funktionen zur Verfügung gestellt:

* Auf- und Abwärts-Zeitgeber
* Zugriffe auf analoge und digitale Prozeßperipherie
* Komplette Reglerblöcke (PI, PID)
* Status- und Kommandofunktionen für die seriellen Kanäle
* Kommunikationsfunktionen

2.2.8 KANALNUMMERN

Für Befehle mit "ON (...)"-Zweig erfolgt E/A auf die der Kanalnummer entsprechende Schnittstelle:

Kanalnummer	logische Belegung unter CP/M	unter PC-DOS
0	CON-Schnittstelle	CON:-Einheit
1	LST-Schnittstelle	LPT:-Einheit
2	AUX-Schnittstelle (CPM 3.0)	COM1:-Schnittstelle
3-9	(konfigurationsabhg.)	
10-xx	Datei-Kanäle	10-19 Dateikanäle

Die freien seriellen Kanäle können nicht nur zum Anschluß von Standardperipherie genutzt werden, sondern eignen sich vorzüglich zur Anbindung komplexerer Schnittstellen wie z.B. IEC-Bus oder von DFÜ-Kanälen.

Details siehe Installationsanleitung für serielle Kanäle.

2.2.9 KOMMANDOS

Die Kommandos im MSRBASIC dienen entweder Editierzwecken (LIST, SAVE, LOAD, NEW) oder Festlegung der Betriebsart (RUN, FREEZE, SYS).

2.2.10 KOMMENTARE

Kommentare werden durch REM-Anweisungen gekennzeichnet.

2.2.11 OPERATOREN

Skalare Operatoren

a) Rechenoperatoren

		Wertigkeit
Potenzoperator	: ^	3
Multiplikation	: *	2
Division	: /	2
Addition	: +	1
Subtraktion	: -	1

b) Vergleichsoperatoren

größer	: >
kleiner	: <
gleich	: =
größer-gleich	: = > oder > =
kleiner-gleich	: = < oder < =
ungleich	: < > oder > <

c) Stringoperationen

Verkettung	: +

Matrix-Operatoren

		Wertigkeit
Multiplikation	: *	2
Element-Multipl.	: .	2
Addition	: +	1
Subtraktion	: -	1

2.2.12 PROGRAMM

MSRBASIC kennt drei Typen von Programmen: Das **Hintergrundprogramm** entspricht dem üblichen BASIC-Programm. Es wird über das RUN-Kommando oder einen unmittelbaren GOTO-Befehl gestartet. Die **Beendigung** des Programms erfolgt über die entsprechenden Anweisungen (RETURN, STOP, END), über Fehlermeldungen oder durch ein Abbruch-Kommando vom Terminal aus (Eingabe von "Ctrl-C" bzw. "Ctrl-P" bei MSR-BASIC-Slave-Rechnern).

Der zweite MSRBASIC-Programmtyp heißt "TASK" und wird zyklisch vom Echtzeitbetriebssystem gestartet (s. nächsten Abschnitt). Sollen alle zyklische Programme nicht mehr gestartet werden, ist "Ctrl-D" (bzw. "Ctrl-T") einzugeben.

Der dritte MSR-BASIC-Programmtyp ist die "SEQuenz". Auch sie wird vom Echtzeitbetriebssystem gestartet (s.u.). Das Anhalten aller SEQuenzen geschieht durch die Eingabe von "Ctrl-A" (bzw. "Ctrl-R").

2.2.13 STARTEN DES INTERPRETERS

Start unter MS-DOS oder CP/M

Der Interpreter wird unter CP/M als MSRxxx.COM-Datei geliefert und unter MS-DOS als MSRxxx.EXE-Datei.

Start MS-DOS oder CP/M (ohne Programm-Laden):

A>MSRxxx

Danach meldet sich der MSR-BASIC-Interpreter mit:

```
MSRBASIC vom xx.xx.198x
Warm- oder Kaltstart (W/K) ?
```

Sofern im BASIC-Programmspeicher ein intaktes Programm vorhanden ist, führt MSR-BASIC automatisch einen Auto-Warmstart durch, d.h. die Anwendervariablen werden nicht gelöscht, das Programm wird an der ersten Zeile ausgeführt.

Hinweis: Die hierbei oft nicht erwünschte anwenderseitige Neuinitialisierung von Parameterwerten durch folgende Anweisungsfolge verhindern:

```
:
IF IniFlg=0 then GOSUB <VarInit>
IniFlg=1
:
```

Bei Kaltstart erfolgt eine Initialisierung der Echtzeituhr, des Echtzeitbetriebssytems und des Arbeitsspeichers, sowie eine **konfigurationsabhängige Kalt-Initalisierung der Prozeß-E/A**. In der Regel bedeutet dies z.B. für die binären Stellsignale ein Schalten in den stromlosen Zustand.

Bei Warmstart wird die Echtzeituhr initialisiert und das ACTIVE-Bit aller TASKS und SEQuenzen zurückgesetzt, um einen definierten Neu- bzw. Wiederstart zu gewährleisten. Außerdem wird eine **konfigurationsabhängige Warm-Initialisierung der Prozeß-E/A** vorgenommen. Dies bedeutet z.B. für die binären Stellsignale, daß die E/A-Bausteine in Ausgaberichtung neu programmiert werden.

Start unter MS-DOS oder CP/M mit Programm-Laden

Beim Aufruf "MSRBAS <Programm-Name>" wird nach dem Erscheinen der MSR-BASIC-Startmeldung das Programm <Programm-Name.BAS> von Diskette geladen (Anzeige "L") und sofort ausgeführt.

stand-alone-Start

In einer stand-alone-Konfiguration gibt es noch zwei weitere Startvarianten, nämlich den Autostart eines in EPROM abgelegten Anwenderprogramms (Eingabe von P) bzw. das automatische Laden eines Programms von einem voreinstellbaren seriellen Kanal und anschließende Starten (Eingabe von "L").

2.2.14 TRANSPARENT-BETRIEB (VIRTUELLES TERMINAL)

MSR-BASIC unterstützt im Rahmen der SAVE-, LOAD- und LIST-Kommandos den Datenverkehr von und zu einem zweiten Rechner (Gastrechner). Bei Ausführung dieser Kommandos geht der Interpreter vorübergehend in den Transparent-Betrieb, d.h. es wird die uneingeschränkte Verbindung zwischen dem Benutzer und dem Gastrechner aufgebaut. Erst die Eingabe eines speziellen Fluchtsymbols (^A bzw. ^R) veranlaßt die Ausführung des Kommandos (s. LOAD-Kommando).

2.2.15 VARIABLE

Ein MSRBASIC-Programm kann maximal 255 verschiedene Variable aufweisen. Es gibt vier Arten von Variablen, nämlich Skalare und Felder jeweils vom Datentyp "REAL" oder "STRING".

Jede Variable besitzt einen Namen, der maximal sechs Zeichen aufweist. Das erste Zeichen ist immer ein Buchstabe. Als folgende Zeichen sind Ziffern, Buchstaben und der Unterstrich "_" erlaubt. Außerdem sind aus Gründen der Kompatibiltät zu Microsoft-BASIC die Zeichen "%" und "#" zugelassen.

Zeichenkettenvariable (Stringvariable) sind durch ein "$"-Zeichen zusätzlich zum Variablennamen gekennzeichnet.

<u>Beispiele:</u> A_min, A0%, Azeich(2), ESC$, X$(1,2)

Die maximale Länge von Stringvariablen wird bei der MSRBASIC-Konfigurierung festgelegt (Voreinstellung: 40 Zeichen). Stringfelder weisen als Elemente Stringvariable auf. Die Namen der Wochentage können z.B. in ein Stringfeld abgespeichert werden:

```
DIM WO$(7)
WO$(1)="Montag"
```

2.2.16 ZAHLENBEREICH

Die größte Zahl, die vom Interpreter verarbeitet wird, ist 4.61168E+18, die kleinste 6.46235E-27. Tritt bei einer arithmetischen Operation ein Überlauf auf, erfolgt eine Fehlermeldung. Dagegen wird bei einem Unterlauf das Ergebnis automatisch zu null gesetzt. Es können Integer-, Festkomma- und Gleitkommazahlen mit und ohne Exponenten eingegeben werden. Es dürfen nicht mehr als sieben Stellen eingegeben werden.

3. EDITOR-FUNKTIONEN

3.1 ÜBERSICHT

In MSR-BASIC stehen bei der Eingabe von Programmzeilen komfortable, **WordStar(R)-orientierte** Editorfunktionen zur Verfügung.

Eine weitere Komfortsteigerung bringt die erweiterte **Syntaxprüfung** bei der Programmzeilen-Eingabe, die nunmehr fehlende Klammern oder Anführungszeichen sofort entdeckt. Damit werden MSR-BASIC-Programme auch sicherer, da syntaxbedingte Laufzeitfehler verringert werden.

3.2 FUNKTIONSBESCHREIBUNG

Bei der Eingabe einer Programmzeile (oder von INPUT aus) ist grundsätzlich der "Einfüge-Modus" in Kraft. Folgende Editierfunktionen stehen zur Verfügung:

Taste	Funktion
BS (=^H)	Cursor eine Position nach links
TAB (=^I)	Cursor eine Position nach rechts
DEL	Zeichen links vom Cursor löschen
^G	Cursor-Zeichen löschen
^X	ganze Zeile löschen
CR	Eingabe beenden (unabhängig von Cursor-Position)
^C (bzw. ^P)	Eingabe abbrechen

Alle anderen Steuerzeichen werden ignoriert.

Beispiel 1: Fehlende "Klammer zu" einfügen

```
    ("_" = Cursorposition)
                                                  ! Fehler
                                                  V
Ausgangssituation:  1210 x1 = x2*(DIN(NOTAus + Hand)_


1. Cursor mit BS auf Fehlerposition fahren:
                    1210 x1 = x2*(DIN(NOTAus_+ Hand)

2. Klammer einsetzen:
                    1210 x1 = x2*(DIN(NOTAus)_+ Hand)

3. Zeile abschicken (CR eingeben)
```

3.3 ÄNDERUNG VORHANDENER PROGRAMMZEILEN

Um bereits vorhandene Programmzeilen mit Hilfe der beschriebenen Editorfunktionen zu ändern, ist die Zeile durch

"# <Zeilennummer>"
bzw. "SAVE <Zeilennummer>"
bzw. "LIST <Zeilennummer>"

anzuwählen.

Hierbei wird die angewählte Zeile ausgelistet, der Cursor bleibt jedoch am Ende der Zeile stehen. Daraufhin kann mit BS der Cursor nach links zur fehlerhaften Stelle bewegt werden.

Beispiel 2: wie oben

Ausgangssituation wird mit "# 1210" (=SAVE 1210) hergestellt:
("_" = Cursorposition)

```
1210 x1 = x2*(DIN(NOTAus + Hand)_
```

Danach Schritte 1...3 wie oben!

Nach Abschluß der Bearbeitung der laufenden Zeile zeigt das LIST-Kommando in dieser Betriebsart die nächste Zeile in der gleichen Weise an. Durch die Eingabe von CR oder Leerzeichen wird die Anzeige ohne Veränderung der aktuellen Zeile weitergeschaltet.

Weitere Hinweise

Syntaxfehler: Werden bei der Programmeingabe Syntaxfehler erkannt, wird die fehlerhafte Zeile nach der Fehlermeldung "Syntax-Error" automatisch im Editiermodus zur Fehlerkorrektur angeboten.

FREEZE-Betrieb: Im Echtzeitbetrieb sind die Editiermöglichkeiten über das LIST-Kommando unterbunden.

4. ECHTZEITSPEZIFISCHE SPRACHMITTEL

4.1 ÜBERSICHT

Um die Echtzeitmöglichkeiten von MSR-BASIC angemessen einsetzen zu können, ist es neben der Kenntnis der Einzelbefehle vor allem notwendig, sich mit dem im folgenden beschriebenen Grundkonzept vertraut zu machen.

4.1.1 MSR-BASIC-ECHTZEITKONZEPT

Die MSR-BASIC-Echtzeit-Philosophie wird deutlich, wenn man Automatisierungsaufgaben bei industriellen Prozessen hinsichtlich ihres Zeitverhaltens unterteilt. Im wesentlichen lassen sich hierbei zwei Grundtypen von Verarbeitungsfunktionen erkennen:

1) immer oder zeitweise im Eingriff befindliche "zeitkontinuierliche" Funktionen wie z.B. lineare Regelungen, Überwachung und Verriegelung, Filterung u.a.m.,

2) zeit- oder prozeßgeführte Ablaufsteuerungen, in denen sich ein ereignisdiskreter Prozeßablauf widerspiegelt.

```
                 +--------------------------------+------------------------------+
                 ! kontinuierliche                ! ereignisorientierte          !
                 ! MSR-Funktionen                 ! MSR-Funktionen               !
+----------------+--------------------------------+------------------------------+
!Beispiele       ! Regelungen                     ! Ablaufsteuerungen            !
!                ! Überwachung-Verriegelung       ! Batch-Prozesse               !
!                ! Filterung                      !                              !
!                !                                !                              !
+----------------+--------------------------------+------------------------------+
!graghische      ! Signalflußplan                 ! Funktionsplan,Petri-Netz     !
!Darstellungs-   !                                !                              !
!form            !                                !            !                 !
!                !                                !            ! +---            !
!                !                                !            ! !               !
!                !        +---------+             !      +-----+-+-+             !
!                !  Xe    !         !  Xa         !      !         !             !
!                !  ---\  !         !---\         !      +---------+             !
!                !      > !         !    >        !            !                 !
!                !  ---/  !         !---/         !            ! +---            !
!                !        !         !             !            ! !               !
!                !        +---------+             !      +-----+-+-+             !
!                !                                !      !         !             !
!                !                                !      +---------+             !
!                !                                !            !                 !
+----------------+--------------------------------+------------------------------+
!MSR-BASIC       ! TASK                           ! SEQuenz                      !
!Programmtyp     !                                !                              !
!                !                                !                              !
+----------------+--------------------------------+------------------------------+
```

Schaubild 4-1: Einteilung der MSR-Funktionen

Diesem Dualismus trägt MSR-BASIC im Gegensatz zu vielen Realzeitsprachen durch das explizite Bereitstellen von zwei nebenläufige Programmtypen Rechnung, nämlich "TASK" für zyklische Programme und "SEQuence" für Ablaufsteuerungs-Programmteile.

Bei reinen Regelungsanwendungen werden nur TASKs verwendet, während z.B. Chargen-Prozesse die Koordination der Regelungsfunktionen durch übergeordnete Ablaufsteuerungen (SEQuenzen) erfordern. Andererseits sind rein ereignis-orientierte (Fertigungs-, Montage- u.a.) Prozesse durch parallele Abläufe gekennzeichnet, die sich durch verbale Spezifikationen, Petri-Netze oder Funktionspläne beschreiben und darstellen lassen. Diese Beschreibungsformen lassen sich sehr leicht in MSR-BASIC-SEQuenzen abbilden.

Die grundlegend unterschiedliche Aufgabenstellung von TASKs und SEQuenzen erklärt auch den unterschiedlichen internen Programmbearbeitungs-Modus: TASKs sollen periodisch gestartet und möglichst in einem Zug abgearbeitet werden, um ihrer quasikontinuierlichen Funktion gerecht zu werden. Ideal wäre für sie eine unendlich kurze Ausführungszeit.

Dagegen hängt die Ausführungsgeschwindigkeit einer SEQuenz kaum von der Rechengeschwindigkeit des Prozessors, sondern vom Zustand des technischen Prozesses ab.

Diese Abhängigkeit der Programmfortsetzung vom Prozeßzustand wird allgemein als "Weiterschaltbedingung" bezeichnet.

Im folgenden Abschnitt wird summarisch erläutert, wie der Anwender die MSR-BASIC-Echtzeitmittel handhabt.

4.1.2 ORGANISATION UND HANDHABUNG DES ECHTZEITBETRIEBS

In MSR-BASIC 4.0 kann der Anwender bis zu 5 zyklische Programme (TASKs) und bis zu 25 Ablaufsteuerungen (SEQuenzen) nebenläufig (quasigleichzeitig) betreiben. Hierzu stehen ihm neben der beschriebenen WAIT-Anweisung als Sprachmittel die Anweisungen DEFINE, START, ACTIVATE und SUSPEND (s.u.) zur Verfügung. Der Echtzeitrachenbetrieb ist nun durch folgende Eigenschaften gekennzeichnet:

- TASKs werden als normale BASIC-Unterprogramme geschrieben und über die DEFINE-Anweisung durch Angabe der Nummer (Priorität), des Aufruf-Zeitintervalls und der Startzeilennummer in die Echtzeitverwaltung integriert.

- SEQuenzen werden ebenfalls als Unterprogramme geschrieben und durch Angabe der Nummer und der Startzeile in die Sequenzverwaltung eingegliedert.

- Die START bzw. ACTIVATE-Anweisung gibt die zyklische Bearbeitung von TASKs bzw. die einmalige Ausführung von SEQuenzen frei.

- Mit der SUSPEND-Anweisung können TASKs und SEQuenzen gezielt aus der Bearbeitung durch das Betriebssystem herausgenommen werden.

- Höher priore TASKs können aktive Programme niederer Priorität (=TASKs, SEQuenzen, Hintergrundprogramm) unterbrechen, wobei das laufende Statement noch bearbeitet wird, um inkonsistente Daten zu vermeiden.

- Die Weiterschaltbedingung in Ablaufsteuerungen (SEQuenzen) wird mit Hilfe der WAIT-Anweisung formuliert. Die integrierte Überwachung der Wartezeit mit der Möglichkeit zur Ausnahmebehandlung ist ein sehr effizientes Mittel, um Fehler im technischen Prozeß zu erkennen und gezielt abzufangen.

- Die Quasinebenläufigkeit der SEQuenzen orientiert sich an den einzelnen Schritten der Ablaufsteuerung. Dies bedeutet, daß eine SEQuenz solange den Prozessor bekommt, bis sie auf ihr nächstes WAIT-Statement stößt, also einen Ablaufschritt ausgeführt hat, oder ein bestimmtes Rechenzeitintervall (10ms) verbraucht hat. Danach bekommt die nächste SEQuenz den Prozessor, bis alle SEQuenzen auf die Erfüllung ihrer momentanen Weiterschaltbedingung warten. Diese werden vom Echtzeitbetriebssystem zyklisch unter Beachtung der Priorität überprüft.

- Werden keine Echtzeitprogramme ausgeführt, steht der MSR-BASIC-Interpreter für Hintergrundprogramme, Kommandos und direkte Anweisungen zur Verfügung.

Da alle Variablen global gültig sind, kann der Benutzer auch ohne spezielles Kommunikationsprogramm während des Echtzeitbetriebs auf alle Variablen mit Hilfe der LET/PRINT-Anweisungen zugreifen!

- Der Datenaustausch zwischen den Echtzeitrechenprozessen geschieht ebenfalls über die global gültigen Variablen.

Damit sind alle Voraussetzungen für einen flexiblen Echtzeitbetrieb erfüllt, wobei MSR-BASIC durch eine Reihe weiterer realzeitorientierter Fehlersuch-Hilfen (SEQLIST, TRACE SEQ, TRACE TASK) von vornherein eine hohe Transparenz gewährleistet.

4.1.3 BEISPIEL: "MESSWERTERFASSUNG"

Als einfaches Beispiel soll eine typische Meßwerterfassungsaufgabe betrachtet werden, bei der die Sprungantwort einer Regelstrecke aufgenommen werden soll.

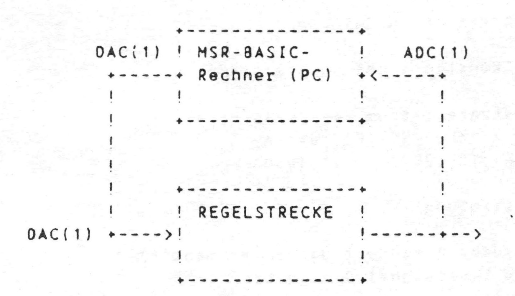


Schaubild 4-2: Konfiguaration "Identifikation einer Regelstrecke"

Diese Aufgabe gliedert sich in mehrere **Phasen**:

Phase 1: Initialisierung von Meßaufbau und Regelstrecke
Phase 2: Aufnahme der Meßwerte
Phase 3: Meßwertweiterverarbeitung (Filtern, Archivieren)

Das im folgenden aufgelistete Programm "MESSDEM.BAS" zeigt eine möglche Implementierung dieser Aufgabe.

Der Initialisierungsteil (Zeilen 100...190) enthält als wesentlichen Teil die Deklaration der verwendeten Echtzeitprogramme (SEQ1, TASK1) und startet SEQ1.

Das Meßwerterfassungs-Hauptprogramm weist genau die oben beschriebene Ablaufstruktur auf und wird deshalb als SEQuenz (Zeilen 1000ff) formuliert.

Entsprechend dem zweigleisigen MSR-BASIC-Echtzeitkonzept ist es in der Phase 2 naheliegend, das zyklische Einlesen, Filtern (PT1) und Ablegen von Meßwerten einer "TASK" zuzuordnen.

Diese TASK wird zu Beginn der Phase 2 von der SEQuenz zur zyklischen Bearbeitung freigegeben (ACTIVATE TASK1) und bei Erfüllung eines bestimmten Kriteriums ("alle n Meßwerte eingelesen") wieder gesperrt.

In der letzten Phase werden die Meßwerte auf einer Datei abgespeichert.

4.1.4 LISTING "MESSWERTERFASSUNG"

```
100   REM **** MESSDEM.BAS *****
105     PRINT "Wieviele Messwerte";
110     INPUT N
120     DIM X(N),TDOWN(1)
130     PRINT "Tastzeit TS in Sekunden";
140     INPUT  TS
145     PRINT "Zeitkonstante des Vorfilters";
150     INPUT  TF
155   REM Filtertastrate ist um Faktor 10 groesser
160     TSF=TS/10, A=EXP(-TSF/TF), B=1-A
170     DEFINE TASK1,TSF,2000, SEQ1,1000
180     START SEQ1
190   STOP Initialisierung

1000  REM ***** Sequenz 1 steuert das Experiment *****
1010  REM Zunaechst Stellsignal 0 ausgeben
1020    DAC(1)=0
1040  WAIT 5 <=0
1050    DAC(1)=1,NZ=N,I=1,TDOWN(1)=TSF
1060    START TASK1
1070    PRINT "Messung gestartet"
1080  WAIT FOR NZ=0
1090    SUSPEND TASK1
1100    PRINT "Messung beendet"
1110    PRINT "Auf welche Datei sollen die Messwerte "
1120    PRINT "abgespeichert werden ";
1130    INPUT FN$
1140    OPEN  FN$,10,0
1150  REM Abspeicherungsschleife
1160    FOR I=1 TO N
1170       PRINT ON (10) X(I) AS 10F3
1180    NEXT I
1190    PRINT "Experiment beendet"
1200  RETURN Sequenz 1

2000  REM ***** TASK1 misst zyklisch den Streckenausgang *****
2010    X=A*X+B*ADC(1)
2020    IF TDOWN(1)>0 THEN RETURN
2040    TDOWN(1)=TS
2050    X(I)=X,I=I+1,NZ=NZ-1
2060  RETURN Task
```

4.2 ECHTZEITBEFEHLE

4.2.1 DEFINE

Aufruf:

```
          +-------+             +---------+
      +-->! <ZN> !--+      +-->! DEFINE !--+
      !   +-------+  V      !   +---------+  V
  --->+              +-->+                   +----->>
      !              A      !   +------+     A
      +--------------+      +-->! DEF !-----+
                                +------+
```

```
                                                 +------------+
          +------+  +-----------+ +---+          ! <Aufruf-   !
       +->! TASK !->! <Nummer> !->! , !->! Intervall>!-+          +---------+
       !  +------+  +-----------+ +---+  +------------+ V  +---+  !<Start- !
  >>+--+                                                .+->! , !->! zeile> !-+-->
    A  !   +-----+  +-----------+                      A  +---+  +---------+ !
    !  +-->! SEQ !->! <Nummer> !-----------------------+                      !
    !      +-----+  +-----------+                                             !
    !                                +---+                                    !
    +--------------------------------! , !<-----------------------------------+
                                     +---+
```

Funktion:

Die angegebenen TASKs bzw.SEQuences werden dem Echtzeitbetriebssystem unter Angabe ihrer Kenndaten (Priorität, Startzeile und ggfs. Aufrufintervall) angemeldet.
Bei TASKs beträgt das Aufrufintervall maximal 2.54s. Alle Zeitangaben werden auf 10ms-Stufung abgerundet.

Beispiele:

```
     1000   DEF TASK 1,T1,2000,TASK 2,2*T1,2200,SEQ 1,3000
bzw. 1000   DEF SEQ NotAus,2000,TASK Regler,t_reg,4000
```

Hinweise:

Soll eine TASK mit einer größeren Abtastzeit als 2.54s aufgerufen werden, bietet MSRBASIC über die Timer-Funktionen TDOWN folgende Lösungsmöglichkeit:

```
 100   REM Initialisierung
 120     DEF TASK 1,2,1000
 130     DIM TDOWN(1)
 140     TA=60, TDOWN(1)=1
 150     ACTIVATE TASK 1
  :

1000   REM TASK 1
1010     IF TDOWN(1)>0 THEN RETURN
1010     TDOWN(1)=TDOWN(1)+TA
1020   REM Ab hier beginnt die TASK eigentlich.
  :            :
```

Diese Lösung bietet die Gewähr, daß selbst bei TASK-Ausführungszeiten >2s keine grob falschen Fehler in den Aufrufintervallen entstehen.

Fehler:

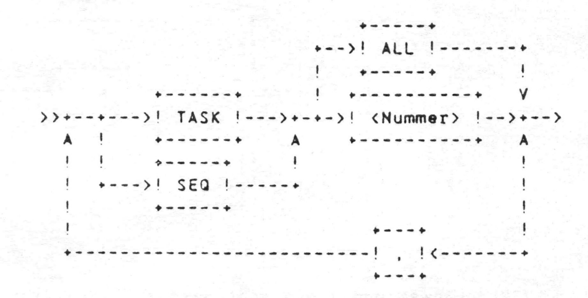
05 TASK bzw. SEQ mit der gewünschten Startzeile existiert nicht

13 Nummer von TASK bzw. SEQ zu groß
oder: Aufrufintervall größer als 2.540 s

4.2.2 START

Aufruf:

```
                +-------+
          +--->! <ZN> !-+
          !     +-------+ V     +--------+
   --->+                  +--->! START !--->>
          !                A     +--------+
          +---------------+
```



Funktion:

Die im Argument der START-Anweisungen angegebenen TASKs werden zur zyklischen Bearbeitung durch das Betriebssystem freigegeben. Die Bearbeitung der angegebenen SEQuenzen wird grundsätzlich bei der **ersten** Programmzeile aufgenommen.

Beispiel:

```
100    START TASK 1,TASK 2,SEQ 1
```

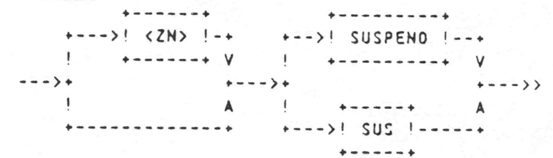
Fehler:

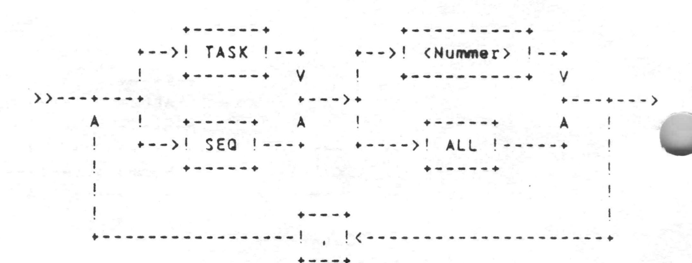
26 TASK bzw. SEQ noch nicht definiert

40 Interpreter nicht im Real-Time-Mode (FREEZE-Mode)

4.2.3 SUSPEND

Aufruf:





Funktion:

Die SUSPEND-Anweisung ermöglicht das Stillegen einzelner oder aller TASKs bzw. SEQuenzen.
Hierbei wird die SEQuenzbearbeitung an der momentanen Stelle (i.d.R. eine WAIT-Anweisung) sofort abgebrochen, wobei evtl. aktivierte Überwachungs-Zeitgeber (WAIT-MAX-Funktion) ebenfalls eingefroren werden, um bei Re-Aktivierung (s.ACTIVATE-Anweisung) eine sachfremde Fehlerbehandlung (...ELSE ...) zu vermeiden.

Die Stillegung von TASKs bedeutet, daß kein Neustart mehr erfolgt, eine laufende TASK jedoch noch zu Ende gebracht wird.

Um bei Programmierfehlern eine schnelle Abbrechmöglichkeit bereitzustellen, können in der Kommandobetriebsart mit der TASK-Stop-Taste alle TASKs und mit der SEQ-Stop-Taste alle Ablaufsteuerungen stillgelegt werden. Sofern dies aufgrund von Endlosschleifen nicht ausreichen sollte, kann über die Stop-Taste ein endgültiger Programmabbruch erzwungen werden.

Beispiel:

```
1120 SUSPEND TASK ALL,SEQ 1
```

Fehler:

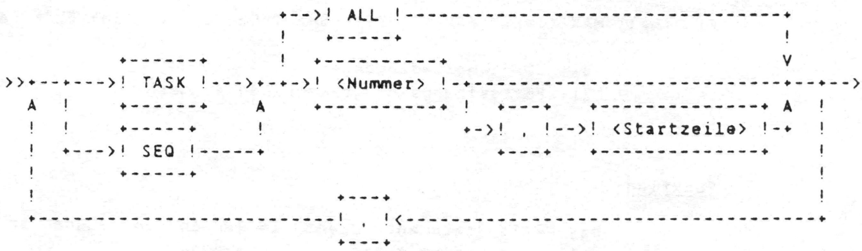
26 TASK bzw. SEQ noch nicht definiert

40 Interpreter nicht im Real-Time-Mode (FREEZE-Mode)

4.2.4 ACTIVATE

Aufruf:

```
       +-------+           +-----------+
   +--->! <ZN> !-+     +--->! ACTIVATE !-+
   !   +-------+ V     !   +-----------+ V
--->+             +--->+                  +--->>
   !              A    !   +------+      A
   +--------------+    +--->! ACT !------+
                           +------+
```



Funktion:

Die im Argument der ACTIVATE-Anweisungen angegebenen TASKs werden zur zyklischen Bearbeitung durch das Betriebssystem freigegeben. Die Bearbeitung der angegebenen SEQuenzen wird bei der "nächsten" Programmzeile aufgenommen (<-> Unterschied zur START-Anweisung).

Dies ist entweder die erste Zeile der SEQuenz, falls sie noch nicht bearbeitet wurde, oder falls sie bereits einmal fertig bearbeitet wurde, also auf eine RETURN-Anweisung gestoßen ist, die darauf folgende.

Falls aber die SEQuenz durch eine SUSPEND-Anweisung (oder durch die TASK-Stop-Taste) abgebrochen worden war, bewirkt die ACTIVATE-Anweisung eine Wiederaufnahme der SEQuenz-Bearbeitung an der Unterbrechungsstelle.

Beispiel:

```
100   ACTIVATE TASK Regler,SEQ NotAus,2050
```

Fehler:

05 gewünschte Startzeile existiert nicht

26 TASK bzw. SEQ noch nicht definiert

40 Interpreter nicht im Real-Time-Mode (FREEZE-Mode)

4.2.5 WAIT

Aufruf:

a) WAIT <Zeit>

b) WAIT FOR <Bedingung>

c) WAIT MAX <Zeit> FOR <Bedingung>

d) WAIT MAX <Zeit> FOR <Bedingung> ELSE <Sprungziel oder Anweisung>

e) WAIT FOR <B1>,...<Bn> THEN <Z1>,...<Zn>

f) WAIT MAX <Zeit> FOR <B1>,...<Bn> THEN <Z1>,...<Zn> ELSE <Anweisung>

Bi: Übergangsbedingungen
Zi: Fortsetzungs-Programmzeilen-Nummern

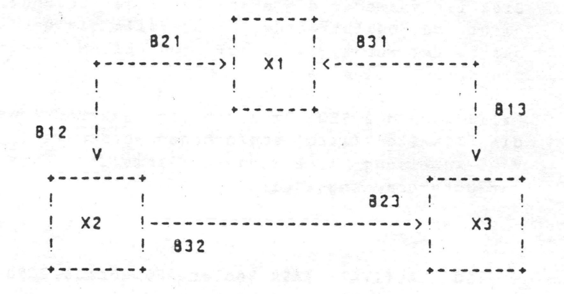
Funktion:

Das WAIT-Statement dient im Rahmen von SEQuenzen zur Formulierung von Weiterschaltbedingungen.

Die WAIT-Anweisung in der Form a) bis d) unterstützt die in der Praxis besonders häufigen linearen Ablaufsteuerungsstrukturen.

Im allgemeinsten Fall kann jedoch eine Ablaufsteuerung aus einem Zustand in mehrere Folgenzustände übergehen.

Hierbei sind Zustandsgraphen ein geeignetes Darstellungsmittel:



Zustandsdiagramm für eine Steuerung mit 3 Zuständen

Dieser Sachverhalt kann mit der Form e) und f) der WAIT-Anweisung formuliert werden.

Hierbei bestimmt bei der internen zyklischen Abprüfung der obigen Weiterschaltbedingung die erste erfüllte Bedingung <Bi> die Zeilennummer <Zi>, bei der die Sequenz fortgesetzt wird.
Ist die Weiterschaltbedingung innerhalb einer vorgebbaren Überwachungszeit (MAX ...) erfüllt, wird wie bei der IF-Anweisung das Programm mit der nächsten Zeile fortgesetzt, ansonsten erfolgt eine programmierbare Fehlerbehandlung, die mit ELSE eingeleitet wird.

<u>HINWEIS:</u>

In Unterprogrammen ist WAIT nicht erlaubt.

<u>Beispiele:</u>

```
zu a) 1000  REM  10 Minuten warten
      1020  WAIT 60*10

zu d) 2000  REM Weiterschaltbedingung mit Zeitüberwachung
      2010  WAIT MAX 100 FOR ADC(1)>X0 ELSE 3000
      :        :
      3000    PRINT "Füllstand nicht rechtzeitig erreicht"
      3010  REM Einlaßventil schließen
      3020    DOUT(1)=0
      3030  STOP

zu d) 4000  REM Boolsche Bedingungen
      4010  WAIT MAX 10 FOR DIN(1)*DIN(6)+NOT(DIN(7)) >= 1 ELSE 1010

zu b) WAIT FOR DIN (Hand) + DIN (NotAUS)
      ist äquivalent zu:
      WAIT FOR DIN (Hand) + DIN (NotAUS) > 0
```

<u>Fehler:</u>

14 Unerlaubte Relation im Bedingungsteil

27 WAIT wurde ausserhalb einer SEQuenz benutzt

39 WAIT in GOSUB-Unterprogramm nicht erlaubt

4.2.6 TRACE

Aufruf:

```
            +-------+          +--------+
        +-->! <ZN> !--+    +-->! TRACE !--+
        !   +-------+  V    !   +--------+  V
   --->+                +--->+               +-->>
        !               A    !    +-----+   A
        +---------------+    +---->! TR !---+
                                  +-----+


                        +-----------+
                     +-->! <Nummer> !--+
                     !   +-----------+  !
          +------+   !                  V
       +-->! SEQ !->+------------------>+--->
       !   +------+                     A
   -->>+                                !
       !     +-------+                  !
       +---->! Task !-------------------+
             +-------+
```

Funktion:

Einschalten des TRACE-Modes, d.h. bei Programmlauf wird jede Zeile auf der Konsole aufgelistet, bevor sie ausgeführt wird. Außerdem wird bei der Ausführung einer INPUT-Anweisung auf einem nicht-interaktiv konfigurierten Kanal die eingegebenen Zeichen auf der Konsole angezeigt.

Folgende Varianten sind möglich:

a) Auslisten aller Befehle (TRACE ohne Zusatz)
b) Auslisten aller SEQuenz-Anweisungen (TRACE SEQ)
c) Auslisten aller TASK-Anweisungen (TRACE TASK)
d) Ein- und Ausschalten einzelner SEQuenzen (TRACE SEQ Misch)

Hinweise:

Die Variante b) des TRACE-Befehls ist insbesondere dann vorteilhaft, wenn man nur den in einer SEQuenz implementierten übergeordneten Ablauf verfolgen will, nicht jedoch die unterlagerten, zyklischen TASK-Module.
Durch die Variante d) läßt sich die TRACE-Funktion auch dann noch sinnvoll nutzen, wenn viele SEQuenzen aktiv sind.
Wird eine SEQuenz im TRACE-Modus ausgelistet, dann erscheint die SEQuenz-Nummer an der linken Seite der Zeile (wie bei SEQLIST).

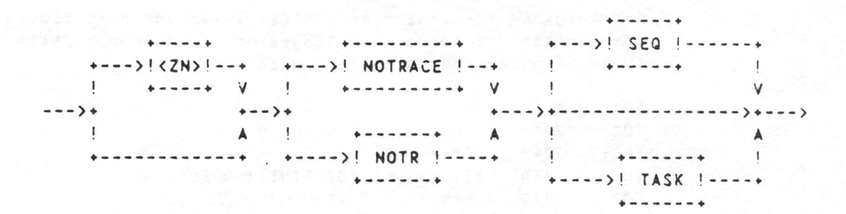
Beispiel:

Der folgende Ausschnitt des TRACE-Laufes von drei SEQuenzen zeigt anschaulich die für diesen Programmtyp wirksame Multitasking-Strategie nach dem round-robin-Prinzip:

```
S01:    1230     DOUT(1)=1
S02:    2210     DOUT(5)=0,DOUT(6)=0
S03:    3240     Presse=0
S01:    1240   WAIT MAX 10 FOR DIN(1)=0 ELSE 4010
S02:    2220     Fix1=1 : Fix2=0
S03:    3250   WAIT MAX 5 FOR DIN(10)=1 ELSE 6120
S02:    2230   WAIT FOR Fix3=0
```

4.2.7 NOTRACE

Aufruf:

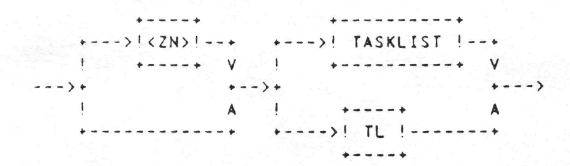


Funktion:

Abschalten der TRACE-Betriebsart global oder für TASKs bzw. SEQuenzen (s.TRACE)

4.2.8 TASKLIST

Aufruf:



Funktion:

Auslisten der Startzeile, der Priorität und des Zustands aller Tasks

Die TASKLIST-Anweisung zeigt in jeder Zeile die Nummer und den Status der ausgelisteten TASKS an.

Beisoiel:

```
T01:          1000  REM Regler
T02:    s     2000  REM Speicher
T03:          3000  .cp54 Filter

 A       A
 !       !
 !       +------Status: s = suspendiert
 !
 +------------TASK-Nummern
```

4.2.9 SEQLIST

Aufruf:

```
               +-------+            +----+                 +-----------+
            +-->! <ZN> !--+      +-->! SL !------+      +--->! <Nummer> !--+
            !   +-------+  V      !   +----+      V      !    +-----------+  V
     --->+                 +--->+                 +--->+                    +--->
            !              A      !               A      !                   A
            +--------------+      !  +----------+ !      +-------------------+
                                  +->! SEQLIST !--+
                                     +----------+
```

Funktion:

Auslisten des momentanen Programmzählerstandes aller SEQuenzen

Die SEQLIST-Anweisung zeigt in jeder Zeile die Nummer und den Status der ausgelisteten SEQUENZ an.

Beispiel:

a)
```
SL
S01:   s      1210 WAIT FOR DIN(HAND)=1
S02:   s      2190 REM ---- ABFAHRSEQUENZ --
S03:   s      3050 WAIT MAX 60 FOR DIN(XSilo5)=0 ELSE 3100

 A     A
 !     !
 !     +------Status: s = suspendiert
 !
 +------------SEQ-Nummer
```

b)
```
SL Fuell
S09:   4070 WAIT MAX 10*60 FOR DIN(xS3max) ELSE 4430
READY
```

Hinweise:

a) Die SEQLIST-Anweisung läßt sich für eine zyklisch aufgerufene Steuerungs-Zustandsübersicht§

b) Das Bedienpersonal kann durch Betätigen einer Taste die SEQ-LIST-Funktion zur Fehlersuche aufrufen; besonders wertvoll ist dies bei der Fehlersuche in gegenseitig verriegelten Ablaufsteuerungsprogrammen.
Liegt zum Beispiel eine Verklemmung vor, d.h. einige SEQuenzen warten - vergeblich - auf die gegenseitige Erfüllung einer Weiterschaltbedingung, so listet die SEQLIST-Funktion die fraglichen WAIT-Anweisungen aus. Durch manuelles Prüfen der Weiterschaltbedingung kann dann in der Regel rasch die Fehlerursache gefunden werden.

4.3 LOGIKFUNKTIONEN

4.3.1 NOT(X)

NOT(X) liefert den Wert "1", wenn X=0 ist, sonst den Wert "0"

4.3.2 BIT(N,X)

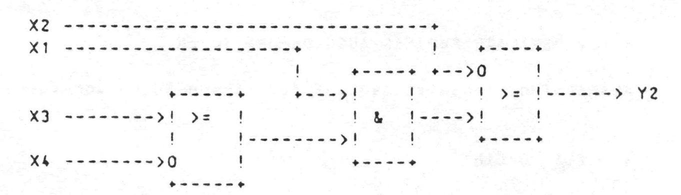
BIT(N,X) wandelt X in eine 16-Bit-Zahl (-32768 <x<32767) und liefert den Wert "1", wenn das N-te Bit (0<=N<=15) gesetzt ist.

4.3.3 BOOLSCHE AUSDRÜCKE

Die Boolschen Operatoren AND und OR können auf die arithmetischen Operatoren * und + zurückgeführt werden.

Beispiel:

a) Funktionsplan



b) MSRBASIC-Formulierung

```
DOUT(2) = DIN(1)*(DIN(3) + NOT(DIN(4)) + NOT(DIN(2))
```

4.4 PROZESS E/A FUNKTIONEN

4.4.1 SKALARE PROZESS-EINGABEFUNKTIONEN

4.4.1.1 ADC(N)

ADC(N) Einlesen eines 16-Bit-Werts vom Analogkanal N
Skalierung ist konfigurationsabhängig

4.4.1.2 DIN(N)

DIN(N) Einlesen des Binärkanals N
Wert beträgt 0 oder 1

4.4.1.3 GET(N)

GET(N) Einlesen des CPU-I/O-Ports N
Wert liegt zwischen 0 und 255 (8-Bit-Version)
bzw. 0 und 64 k (16-Bit-Version)

4.4.1.4 CNT(N)

CNT(N) Einlesen eines 16 Bit Zählerkanals

4.4.2 SKALARE PROZESS-AUSGABEFUNKTIONEN

Ausgabefunktionen stehen auf der linken Seite von Zuweisungen.

4.4.2.1 DAC(N)

DAC(N) Ausgabe auf Stellkanal N (16 Bit genau)
Skalierung ist konfigurationsabhängig

4.4.2.2 DOUT(N)

DOUT(N) Setzen des Binärkanals N
Wert beträgt 0 oder Nicht-Null

4.4.2.3 PUT(N)

PUT(N) Ausgabe einer 8-Bit-Zahl (Wert 0 ... 255)
auf CPU-I/O-Port N
Wert liegt zwischen 0 und 255 (8-Bit-Version)
bzw. 0 und 64 k (16-Bit-Version)

4.4.2.4 CNT(N)

CNT(N) Setzen eines 16 Bit Zählerkanals

4.4.3 VEKTOR-E/A-FUNKTIONEN

Vektorfunktionen werden im Rahmen von VEC-Anweisungen ausgeführt (s.d.)

4.4.3.1 MADC(MO)

MADC(MO) Einlesen eines Meßvektors (Komponenten des Meßortvektors MO legen die einzelnen Eingabekanäle fest)

4.4.3.2 MDAC(SO)

MDAC(SO) Ausgabe eines Stellvektors (Komponenten des Stellortsvektors SO legen die einzelnen Ausgabekanäle fest)

4.4.3.3 MDIN(MD)

MDIN(MD) Einlesen eines Binärvektors

4.4.3.4 MDOUT(SD)

MDOUT(SD) Ausgabe eines binären Steuervektors

<u>Beispiel:</u>

```
 150   DIM XE(N), MO(N), Y(N), SO(N)
  :     :
1000   VEC XE=MADC(MO), MDAC(SO)=Y
```

ist äquivalent zu:

```
1000   FOR I=1 TO N
1010      XE(I) = ADC(MO(I)), DAC(SO(I))=Y(I)
1020   NEXT I
```

4.4.4 ABFRAGE DER PROZESSKONFIGURATION

4.4.4.1 NDAC

NDAC   r liefert die Anzahl der installierten DAC-Kanäle

4.4.4.2 NADC

NADC   r liefert die Anzahl der installierten ADC-Kanäle

4.4.4.3 NDOUT

NDOUT   r liefert die Anzahl der installierten DOUT-Ports (8 Kanäle/Port)

4.4.4.4 NDIN

NDIN   r liefert die Anzahl der installierten DIN-Ports (8 Kanäle/Port)

4.4.4.5 NCNT

NCNT: r liefert die Anzahl der installierten CNT-Kanäle

<u>Beispiel:</u>

```
For i=1 to 8*NDIN
      Print DIN(i) as i;
Next i
```

4.5 ZEITGEBERFUNKTIONEN

4.5.1 TDOWN, TUP

Die Zeitgeber TDOWN (aufwärtszählend) und TUP (abwärtszählend), werden wie eindimensionale numerische Felder behandelt.

a) Deklarieren:

DIM TUP(3),TDOWN(1)	3 Aufwärts-, 1 Abwärtszeitzähler deklarieren

b) Verändern:

TDOWN(1)=100	100 s vorbesetzten

c) Lesen:

PRINT TUP(2)	Momentanstand von Aufwärtszeitgeber 2 ausdrucken

Der Zeittakt beträgt 1s (Voreinstellung).

4.5.2 TIME$

Die Variable TIME$ enthält bei Lesezugriff die Tageszeit in der Form:

"hh:mm:ss" (hh=Stunden, mm=Minuten, ss=Sekunden).

4.5.3 DATE$

Die Variable DATE$ enthält bei Lesezugriff das Datum in der Form:

"dd:mm:aa" (dd=Tage, mm=Monate, aa=Jahre)

4.5.4 DOW$

Die Variable DOW$ gibt die englische Kurzform des aktuellen Wochentages wieder:

SUN	Sonntag	THU	Donnerstag
MON	Montag	FRI	Freitag
TUE	Dienstag	SAT	Samstag
WED	Mittwoch		

Hinweis: Voraussetzung ist eine Hardware-Uhr und ihre Anbindung an MSRBASIC durch den Installateur.

4.5.5 TIME-FACTOR

Ein Lesezugriff auf TIME_FACTOR liefert das aktuelle Aufrufintervall der MSR-BASIC-internen Zeitgeber-Task in 10ms-Einheiten.

In der Regel hat TIME_FACTOR den Wert 100(=1s Zeitauflösung).

In einzelnen Fällen kann es sinnvoll sein, die Zeitinkremente zu verändern (insbesondere zu verkürzen). Eine Auflösung von 100ms erreicht man durch die Anweisung:

```
TIME_FACTOR = 10
```

Diese Beschleunigung wirkt sich aus auf:

- TUP- /TDOWN-Felder
- Wartezeit in WAIT/WAIT MAX-Anweisung

Sie wirkt sich nicht aus auf TASK-Aufrufintervalle !!

4.6 REGLERFUNKTIONEN

4.6.1 ALLGEMEINES

Als parametrierbare Reglerfunktionsblöcke werden in MSR-BASIC (skalare und vektorwertige) PI- und PID-Regelalgorithmen mit Positions- und Geschwindigkeits-Ausgang angeboten. Sie realisieren folgende Grundformel:

Berechnen des Zuwachses:

```
Dyi := Kp*(ei-ei-1+KI*ei+KD*(ei-2*ei-1+ei-2))

                    A<- nur bei PID -->A
```

Umspeichern der vergangenen Regeldifferenzen:

```
ei-2 := ei-1, ei-1 := ei
```

Die Stellungsversion summiert dazu noch die Stellinkremente auf: yi := yi-1 + Dyi

Der MSRBASIC-PID-Funktionsblock entspricht folgendem BASIC-Programm:

```
1000  REM PID-Algorithmus
1010  REM Bedingte Bildung des Integralteils
1020    IF AK*E<=0 THEN I=KI*E ELSE I=0
1030    DY = KP*(E-E1 + I + KV*(E-2*E1+E2))
1040    AK = AK+DY
1050    E2 = E1, E1 = E
1060    DY = AK
1070  REM Stellsignal begrenzen
1080    IF DY>0 THEN GOSUB 1100 ELSE GOSUB 1150
1090    AK = AK-DY, Y = Y+DY
1099  RETURN

1100  REM positives Stellinkrement begrenzen
1110    IF DY>YD THEN DY=YD
1120    IF DY>YO-Y THEN DY=YO-Y
1130    IF DY>YU-Y THEN DY=YU-Y
1140  RETURN

1150  REM negatives Stellinkrement begrenzen
1160    IF DY<-YD THEN DY=YD
1170    IF DY<YU-Y THEN DY=YU-Y
1180    IF DY>YO-Y THEN DY=YO-Y
1190  RETURN
```

Über den reinen PID-Algorithmus hinaus (Zeile 1030 und 1090:2) sind also folgende wichtige Funktionen realisiert:

* Stoßfreie on-line-Parameterverstellung durch interne Geschwindigkeitsform

* Automatisches Abschalten des I-Anteils (Zeile 1020), wenn eine Stellgrößen- oder Stellgeschwindigkeitsbeschränkung (Zeilen 1100 bis 1190) erreicht wird (Verhinderung der Integralsättigung).

* Verhinderung des proportional-differentialen Anfahreffekt bei den Versionen mit Geschwindigkeitsausgang durch die Zwischenspeichermethode (1040,1060,1090)

* Automatische Stellgrößenanpassung an dynamisch veränderte Grenzwerte YO bzw. YU (1130,1180) (wichtig bei Kaskadenschaltung !)

Es ist allerdings zu beachten, daß der I-Anteil (KI) niemals zu null gesetzt werden darf, da sonst große stationäre Regeldifferenzen auftreten (Integraler Offset).

4.6.2 P/PID-EINZELREGLER

Die Reglerfunktionen können syntaktisch wie Standardfunktionen verwendet werden, also auch in arithmetische Ausdrücke eingebunden werden. Als Ausgangsgröße wird je nach Algorithmus der Momentanwert der Stellgröße oder der Stellgeschwindigkeit geliefert.

```
PID-Stellungsalgorithmus:........  PID(Y,E,E1,E2,KP,KD,KI,YO,YU,YD,AK)
PID-Geschwindigkeitsalgorithmus:. GPID(Y,E,E1,E2,KP,KD,KI,YO,YU,YD,AK)
PI-Stellungsalgorithmus:.........   PI(Y,E,E1,KP,KI,YO,YU,YD,AK)
PI-Geschwindigkeitsalgorithmus:..  GPI(Y,E,E1,KP,KI,YO,YU,YD,AK)
```

Hierbei bedeuten:

```
Y : Stellgröße
E : momentane Regeldifferenz      (Eingangsgröße)
E1: letzte Regeldifferenz         (Zustandsgröße)
E2: vorletzte Regeldifferenz      (Zustandsgröße)
AK: Zwischenspeicher              (Zustandsgröße)
KP: Verstärkung
KD: Differentialbeiwert           (Regler-Parameter)
KI: Integralbeiwert
YO: Stellgrößenobergrenze
YU: Stellgrößenuntergrenze        (Parameter)
YD: Max. Stellgrößenzuwachs
```

Zu beachten ist, daß die Reglerfunktionen neben dem Stellsignal als eigentlicher Ausgangsgröße auch das Umspeichern der früheren Werte der Regeldifferenz und die Verwaltung des Zwischenspeichers vornehmen. Der Anwender muß hierfür nur die entsprechenden Variablen bereitstellen.

4.6.3 VEKTOR P/PID-REGLER

Im Rahmen des VEC-Statements können neben den Prozeß-E/A-Funktionen MADC, MDIN, MDAC, MDOUT auch die Reglerblöcke PI und PID verwendet werden. Eine Anweisung:

```
1000    VEC  Y = PI(Y,E,E1,KP,KI,YO,YU,YD,AK)
```

ist funktionell gleichbedeutend mit:

```
1000    FOR  I=1 TO N
1010       Y(I) = PI(Y(I),E(I),E1(I),KP(I),KI(I),YO(I),
                  YU(I),YD(I),AK(I)
1020    NEXT I
```

Allerdings halbiert sich bei Verwendung der VEC-Anweisung sowohl der Schreib- wie der Rechenaufwand.

5. EIN/AUSGABE-SPRACHMITTEL

5.1 ÜBERSICHT

Ein Hauptvorteil von MSRBASIC ist seine Fähigkeit, auch im Multitask-Betrieb unmittelbare Kommandoeingaben zu verarbeiten.

Die im folgenden beschriebenen MSR-Features verbessern diesen Ansatz ganz wesentlich durch

- **Fenstertechnik** zur automatischen Trennung der Kommando-E/A-Vorgänge von programmgesteuerten Bildschirmzugriffen (Prozeßdarstellung, Pro tokollierung etc.)

- Terminal-unabhängige **Cursor-Positionierung** im Rahmen der PRINT-An weisung

- Schneller Zugriff auf als Files abgelegte **Bildschirmmasken**

- **Beschleunigung** der Kommando-Bedienung auch bei hoher Echtzeitbe-lastung

- Einfache Gestaltung von **Steuerungs-Übersichtsbildern** durch verbes serte SEQLIST-Programmanweisung

- Erweiterung der TRACE-Anweisung, um gezielt einzelne SEQuen-zen zu verfolgen.

5.1.1 FENSTERTECHNIK

Die MSR-Fenstertechnik erleichtert in hohem Maß die aufwandsarme Implementierung von CRT-basierenden Bedien- und Anzeigefunktionen mit MSRBASIC, wobei insbesondere die residenten Echtzeiteingriffsmöglichkeiten von MSRBASIC unterstützt werden.

Der Bildschirm wird hierbei in ein oberes und ein unteres Fenster aufgeteilt.

Das **obere Fenster** ("Anwenderfenster") steht dem MSR-Anwenderprogramm für die Ausgabe mit Hilfe von PRINT-Anweisungen zur Verfügung.

Wie die beiliegenden Beispiele zeigen, lassen sich in diesem Fenster übliche Darstellungsformen der Prozeßleittechnik wie z.B. Balkendiagramme oder Trendkurven implementieren. Insbesondere unterstützt die THERMOPLAN-Fenstertechnik den situationsbedingten Wechsel verschiedener Prozeßdarstellungen.

Das **untere Fenster** erlaubt Zugriffe auf den MSRBASIC-Kommando-Interpreter.

Dieses Fenster ("Kommandofenster") wird aktiviert bei:

- MSR-BASIC-Kommandos (z.B. LIST, SEQLIST etc.),
- direkten Anweisungen (z.B. PRINT Y, W=50 etc.),
- Fehlermeldungen
- TRACE-Ausgaben

Die MSR-Fenstertechnik stellt softwaretechnisch ein **Ausgabefilter** dar, das bei jeder Ausgabeanforderung des MSRBASIC-Interpreters den momentanen, internen Status des Interpreters auswertet, um das aktuelle Zeichen in das richtige Fenster zu positionieren.

Diese Methode garantiert eine absolute Unabhängigkeit der Anwenderprogrammierung von der Fensterverwaltung (Transparenz).

5.1.2 PROGRAMMIERUNG

Die Fenstertechnik wird aktiviert und desaktiviert über die MSR-BASIC Systemvariable UPPER_SCREEN.

<u>Window Einschalten:</u>

Aufruf: <NNNN> UPPER_SCREEN = NWIN ~~+ 256~~

NWIN: Zeilenanzahl des oberen Fensters (0 < NWIN< 23)

Wirkung:
- Bildschirm löschen
- MSR-Fensterfunktion aktivieren

<u>Window Ausschalten:</u>

Aufruf: UPPER_SCREEN = 0 bzw. CLS

Wirkung:
- Bildschirm löschen
- MSR-Fensterfunktion ausschalten

<u>Cursorpositionierung:</u>

Aufruf : PRINT [Zeile,Spalte] <Zeichen>

Wirkung:
- Cursor auf Position Zeile,Spalte setzen
- <Zeichen> ausgeben

<u>Masken-Dateien ausgeben:</u>

Aufruf:
```
OPEN     <Maskenname>,<Log.Einheit>,1
COMMAND  (Log.Einheit)=1
CLOSE    <Log. Einheit>
```

Wirkung:
- Datei <Maskenname> wird auf den Bildschirm ausgegeben.

Abfragen des oberen Fensters:

Aufruf: print UPPER_SCREEN

BEISPIEL: Demoprogramm WINDEM.BAS
========

Das Programm WINDEM.BAS ist als einfachstes Nicht-Echtzeit-Beispiel konzipiert.

Zunächst wird in den Zeilen 110-130 das LIST-File des Demoprogramms ("WOLIS.BAS") quasi als Bildschirm-Maske ausgeworfen (sofern die Datei vorhanden ist).

Nach einer Wartezeit wird der Bildschirm geteilt (Z.140). In der Folge wird gezeigt, wie programmgesteuerte Ausgaben automatisch in das obere Fenster gelangen, während die TRACE-Anzeige in das untere Fenster gelenkt wird.

```
100   REM ---------------- WINDEM.BAS ------------------
102   REM  01.06.88
104   REM
105     CLS
110     OPEN "WOLIS.BAS",10,1
120     IF BIT(7,STATUS(10))=1 THEN COMMAND(10)=1
130     CLOSE 10
135     FOR I=1 TO 1000
136     NEXT I
140     UPPER_SCREEN = 8
150     TRACE
160     PRINT [0,10] " MSR-Zweifenstertechnik "
200     FOR I=1 TO 20
210         PRINT CHR$(I + 64);
220     NEXT I
230     NOTRACE
240   STOP
```

Am Bildschirm ergibt sich folgendes Bild:

```
                 +---------------------------------------------------------+
Programm-     0 !                 MSR-Zweifenstertechnik                   !
Fenster         ! ABCDEFGH                                                 !
------------->8
              9 !                                                          !
Kommando-       !   210        PRINT CHR$(I+64);                           !
Fenster         !   220     NEXT I                                         !
                !   210        PRINT CHR$(I+64);                           !
                !   220     NEXT I                                         !
                !   210        PRINT CHR$(I+64);                           !
                !   220     NEXT I                                         !
             23 !                                                          !
                 +---------------------------------------------------------+
```

5.2 E/A-ANWEISUNGEN

5.2.1 INPUT

<u>Aufruf:</u>

```
          +-------+             +--------+             +-----+ +----------+
      +-->! <ZN> !--+      +-->! INPUT !--+       +-->! ON !--! <Kanal> !--+
      !   +-------+  V      !   +--------+  V       !   +-----+ +----------+  V
  --->+              +--->+                 +--->+                            +-->>
      !              A      !    +----+     A       !                            A
      +--------------+      +--->! I !-----+       +----------------------------+
                                 +----+


            +-------------+
  -->>+--->! <Variable> !->+--->
      A    +-------------+  !
      !       +----+        !
      +------! , !<--------+
              +----+
```

<u>Funktion:</u>

Das INPUT-Statement ermöglicht das programmgesteuerte Einlesen alphanumerischer Daten von

- der Konsole (Nr. 0)
- einem seriellen E/A-Kanal (INPUT ON(1...9))
- einem Datei-Kanal (INPUT ON(10...19))

Die Daten-Kanäle müssen vorher mit OPEN geöffnet worden sein.

Bei der Ausführung des INPUT-Statements wird grundsätzlich auf den Eingabekanal ein Fragezeichen ausgegeben und die Eingabe geechot, wenn er als interaktiver Eingabekanal installiert wurde (s. Installationsanweisung). Beim Einlesen von einem als nicht interaktiv in stallierten Kanal wird das Echo unterdrückt. In der TRACE- Betriebsart werden allerdings die übersandten Zeichen zur Kon trolle auf der Bedienkonsole ausgegeben.

Sofern numerische Daten eingelesen werden, müssen sie durch ","- oder " "-Zeichen getrennt werden.

Beim Einlesen einer Stringvariablen werden führende Leerzeichen verworfen und nur das "," als Trennzeichen gewertet.

Bei fehlerhafter oder unvollständiger Eingabe wird über eine unmittelbare Abfrage von Bit 3 im Kanal-STATUS-Wort der Eingabezustand überprüft. Die INPUT-Anweisung ist grundsätzlich TIMEOUT gesichert, d.h. wird eine Eingabe nicht innerhalb 20s abgeschlossen, gilt die Eingabe als fehlerhaft.

```
INPUT ON(S1vLU) A$
IF BIT(InpErr) THEN...
```

Die Eingabe und damit das Programm kann unabhängig von der angewählten Einleseeinheit von der Kommandokonsole mit der Stop-Taste abgebrochen werden.

Beispiel:

```
INPUT  A,B,A$
? 1,2,ENDE
READY
PRINT A,B,A$
1.0000E 00  2.00000E 00 ENDE
READY
```

Hinweise bei Echtzeitbetrieb:

Bei Multitaskbetrieb ist die Benutzung des INPUT-Statements mit Vorsicht zu behandeln. Zwar kann es entgegen den sonstigen Regeln von höherprioren TASKs unterbrochen werden, es darf aber im unterbrechenden Programm kein neues INPUT-Statement aufgesetzt werden.

Wenn derartige Konkurrenzfälle nicht ausgeschlossen werden können, empfiehlt es sich, die INPUT-Anweisung nur im Hintergrundprogramm zu verwenden und Einlesevorgänge in Echtzeitrechenprozessen über die INCHAR$-Funktion zu formulieren (s.d.!).

Fehler:

08 Fehlerhafte Variable in Einleseliste

41 Mehr als eine INPUT-Anweisung aktiv

58 Lesen von als write-only eröffneter Datei

5.2.2 PRINT

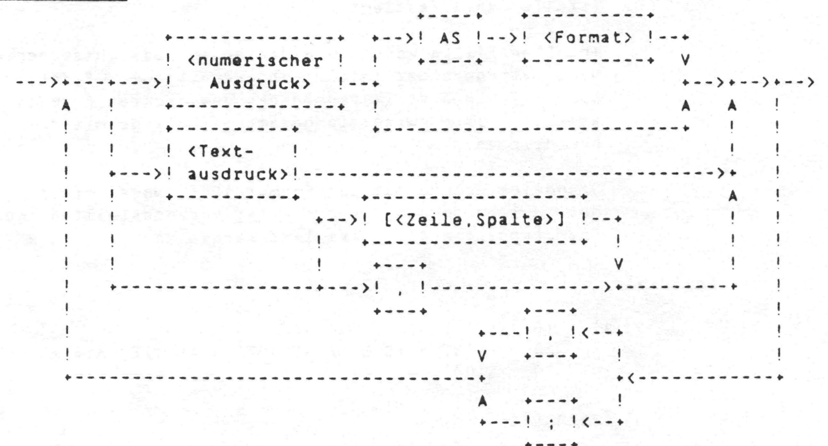
Aufruf:

```
          +-------+          +--------+      +-----------------------+
  +-->! <ZN> !--+   +-->! PRINT !--+   +-->! ON ( <Kanalnummer> ) !--+
  !   +-------+  V   !  +--------+  V   !  +-----------------------+  V
--->+            +-->+              +-->+                             +-->>
  !              A   !    +---+     A   !                             A
  +--------------+   +---->! P !----+   +-----------------------------+
                          +---+

            +----------------+
        +-->! <Druckoperand> !--+
        !   +----------------+  V
   --->>+                       +--->
        !                       A
        +-----------------------+
```

Funktion:

Die PRINT-Anweisung ermöglicht die Ausgabe von beliebigen Zeichenfolgen und Zahlenwerten auf dem Bildschirm oder einer frei wählbaren Ein-/Ausgabe-Einheit (Benutzung des ON-Zweiges). Die Tabellierung kann mit Hilfe eines oder mehrerer Kommas zwischen den auszugebenden Daten vorgenommen werden. Die Formatierung von Zahlenwerten erfolgt über die AS-Angabe.
Die logischen Einheiten 10-19 erlauben Ausgabe auf Diskettendatei. Diese muß vorher mit OPEN im richtigen Mode eröffnet und, falls nicht vorhanden, mit CREATE angelegt worden sein.

Druckoperand:



Tabellierung:

Bei der voreingestellten Tabellierung wird eine Ausgabezeile in 5 Felder mit einer Länge von 13 Zeichen eingeteilt. Die Verwendung des Kommas als Trennungszeichen bewirkt ein Vorrücken der Druckposition zum Beginn des nächsten 13er Feldes. Soll dies verhindert werden, muß der Strichpunkt als Trennungszeichen verwendet werden. Wenn der automatische Zeilenvorschub bei Beendigung des PRINT-Statements unterbunden werden soll, muß als letztes Zeichen ein Komma oder Strichpunkt gesetzt werden.

Cursor-Positionierung

Nach jedem ";"-Zeichen kann eine Cursor-Positionierung vorgenommen werden.

Formatierung:

Folgende Ausgabeformate stehen zur Verfügung:

1. E-Format (Gleitkomma)

Das Format hat die Form fEk bzw. (OfEk), wobei f die Feldlänge (Gesamtzahl der Zeichen mit Vorzeichen und Punkt) und k die Anzahl der Nachkommastellen der Mantisse angibt.

2. F-Format (Festkomma)

Das F-Format hat die Form fFk bzw. (OfFk), wobei f die Feldlänge und k die Nachkommastellen (ohne Punkt) festlegt.

3. I-Format (Ganzzahl)

Das I-Format hat die Form fI bzw. (OfI), wobei f die Feldlänge (Stellenzahl) festlegt.

In allen Fällen werden die Zahlen auf das entsprechende Ausgabeformat gerundet (nicht abgeschnitten). Sofern der auszugebende Zahlenwert im Rahmen des gewünschten Formats nicht darstellbar ist, wird die gesamte Feldlänge mit dem "*"-Zeichen beschrieben.

Grundeinstellung ist das Format 12E4, wobei die Ausgabe führender Nullen durch Angabe einer vorangestellten Null in der Formatspezifikation veranlaßt werden kann.

Beispiele:

a)
```
10   A=7.78
20   PRINT A AS 5I,A AS 10F3,A AS 12E2,A,A AS 4F3
30   STOP
```

Ergebnis:

```
8            7.780        7.78E 00  7.7800E 00     ****
```

b)

```
10      PRINT ON(plu) "PRINTST"
20    REM
30      a=1, b=-1
100     PRINT ON(plu) "Integer-Format"
110     PRINT ON(plu) a AS 6I, b AS 6I
120     PRINT ON(plu) a AS 06I, b AS 06I
200     PRINT ON(plu) "Festkomma-Format"
210     PRINT ON(plu) a AS 9F4, b AS 9F4
220     PRINT ON(plu) a AS 09F4, b AS 09F4
300     PRINT ON(plu) "Gleitkomma-Format"
310     PRINT ON(plu) a AS 9E2, b AS 9E2
320     PRINT ON(plu) a AS 09E2, b AS 09E2
999   STOP
```

Ergebnis:

```
PRINTST
Integer-Format
     1             -1
000001         -00001

Festkomma-Format
   1.0000       -1.0000
0001.0000      -001.0000

Gleitkomma-Format
 1.00E 00      -1.00E 00
01.00E 00      -1.00E 00
```

Fehler:

08 Syntaktisch falsche Form

59 Schreiben auf als read-only eröffnete Datei

5.3 SCHNITTSTELLENFUNKTIONEN

5.3.1 EOF(LU)

Die EOF-Funktion hat den Wert 0, wenn beim letzten Zugriff auf den seriellen Einlese-Kanal LU vom Betriebssystem keine EOF-Bedingung zurückgemeldet wurde bzw. wenn das nächste Einlesezeichen kein ^Z (CP/M-EOF-Zeichen) ist. Eine EOF-Bedingung kann damit als "Kein weiteres Eingabe-Zeichen vorhanden" interpretiert werden und ist in dieser Form (unter CP/M) werksseitig für den CON:-Kanal wie für die Datei-Kanäle implementiert.

Beispiel: "Auf Zeicheneingabe warten"

```
1180  WAIT MAX 10 FOR EOF(0)=0
```

5.3.2 INCHAR$(LU)

Die INCHAR$-Funktion liest - falls vorhanden - ein Zeichen von der durch den Wert von LU bestimmten logischen Einheit ein. Falls kein Zeichen anliegt, wird ein Nullstring übergeben. Man beachte, daß INCHAR$ kein Echo auf den Eingabekanal auslöst.

Die INCHAR$-Funktion ist dann vorteilhaft, wenn man Fehleingaben des Bedieners sofort und gezielt abfangen will.

5.3.3 STATUS(LU)

Die STATUS-Funktion liefert in Form eines Zahlenwertes den momentanen Zustand des seriellen Kanals LU. Dieser Zahlenwert wird vom Betriebssystem in Form einer 16Bit-Zahl übergeben. Hierbei sind nur die Teile der unteren 8-Bit werksseitig belegt und zwar:

Bit 7: =1, wenn LU geöffnet ist
Bit 6: =1, wenn LU als Write-only-Kanal konfiguriert ist
Bit 5: =1, wenn EOF-Bedingung aufgetreten ist (=EOF(LU))
Bit 4: (reserviert)
Bit 3: nach INPUT-Anweisung: 1-TIMEOUT bei INPUT
Bit 2: (reserviert)
Bit 1: =1, wenn ein Zeichen ausgegeben werden kann
Bit 0: =1, wenn ein Zeichen zum Einlesen bereit ist

Beispiel: "Warten auf Eingabe mit Timeout"

```
1190  WAIT MAX 10 FOR BIT
```

5.3.4 COMMAND(LU)

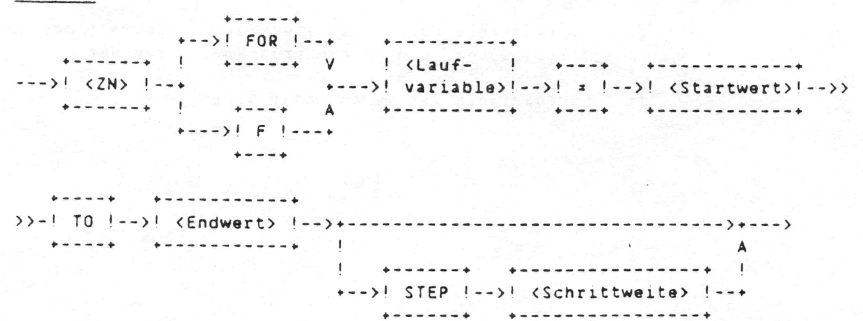
Die COMMAND-Funktion stellt das Gegenstück zur STATUS-Funktion dar. Sie liefert an das Betriebssystem einen vom Benutzer vorgebbaren 16-Bit-Wert, der als Steuerwort zu interpretieren ist.

6. STANDARDSPRACHMITTEL

6.1 PROGRAMMFLUSSTEUERUNG

6.1.1 FOR

Aufruf:



Funktion:

Die FOR-Anweisung erlaubt den Aufbau von Laufschleifen. Als Laufvariable darf nur eine skalare numerische Variable verwendet werden, während Startwert, Endwert und Schrittweite durch arithmetische Ausdrücke vorgegeben werden können (alle Werte erlaubt). Falls eine explizite Schrittweitenangabe fehlt, wird als Schrittweite "1" genommen.

Hinweise:

Zu beachten ist, daß die Laufschleife wenigstens einmal durchlaufen wird, da die Abbruchbedingung erst im NEXT-Statement geprüft wird. Weiterhin sollte man nicht aus einer Laufschleife direkt springen, sondern bei Erfüllung einer zweiten Abbruchbedingung die Laufvariable auf den Endwert setzen (sonst evt. Schwierigkeiten mit Stack für Laufvariable).

Maximal 10 Laufschleifen können ineinander verschachtelt werden.

Die Verwendung von FOR/NEXT-Schleifen in TASKs und SEQuenzen ist möglich, allerdings müssen unterschiedliche Laufvariable verwendet werden (sonst Fehlermeldung 20).

Beispiel:

```
1000  FOR I=1 TO N STEP 2
1020      FOR J=1 TO M
1030          FI=I,FJ=J
1040          IF  A(I,J) >10000 THEN I=N,J=M
1050      NEXT J
1060  NEXT I
```

```
1070  IF FJ<M THEN PRINT "Treffer !"
```

Fehler:

16 "="-Zeichen fehlt

17 "TO" oder "STEP" fehlt

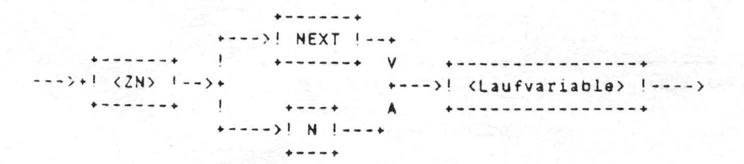
18 Mehr als 10 verschachtelte Laufschleifen

20 Laufvariable bereits in (äußerer) Schleife (oder in anderer TASK, SEQuenz, Hintergrundprogramm) verwendet

21 Laufvariable ist Vektor oder String

6.1.2 NEXT

Aufruf:



Funktion:

Die NEXT-Anweisung bestimmt das Ende einer Laufschleife. Hierzu wird grundsätzlich die Laufvariable um die Schrittweite erhöht und die Schleife abgebrochen, wenn der neue Wert der Laufvariablen über der Grenze liegt. D.h., daß nach Beendigung der Schleife die Laufvariable größer als der Endwert ist.

Beispiel:

Siehe FOR-Statement

Fehler:

19 NEXT wird vor FOR bearbeitet

20 Laufvariable in FOR und NEXT nicht identisch

6.1.3 GOSUB

Aufruf:

```
                    +--------+
               +--->! GOSUB !---+
     +-------+ !    +--------+   V   +-------+
--->! <ZN> !-->+                 +-->! <ZN> !--->
     +-------+ !    +-----+      A   +-------+
               +--->! GS !-------+
                    +-----+
```

Funktion:

Mit dem GOSUB-Statement werden BASIC-Unterprogramme aufgerufen. Aus dem Unterprogramm muß über das RETURN-Statement in das aufrufende Programm zurückgekehrt werden (Verschachtelungstiefe beliebig).

Beispiel:

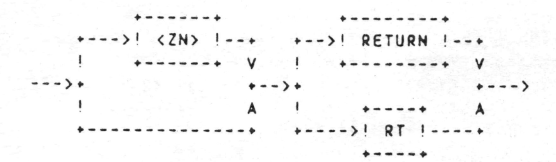
```
1000   GOSUB  2000

2000   RETURN
```

Fehler:

05 Unterprogramm mit der gewünschten Nummer existert nicht

6.1.4 RETURN

Aufruf:



Funktion:

Die RETURN-Anweisung muß am Ende eines über GOSUB aufgerufenen Unterprogrammes, einer TASK oder einer SEQ stehen. Bei der Ausführung wird dann das übergeordnete Programm nach dem GOSUB-Statement fortgesetzt.

Beispiel:

```
      :
1000    GOSUB  2000
      :
2000   LET TA=TA-1
2010   IF TA>0 THEN RETURN
2020   LET TA=T0
2030 RETURN
```

Hinweise:

Bei Beendigung einer SEQuenz durch RETURN wird automatisch die Startzeile der SEQuenz (s. DEFINE-Anweisung) als nächste auszuführende Anweisung im Sequenz-Steuerblock festgelagt, bevor die SEQuenz stillgelegt (suspendiert) wird.

Fehler:

(KEINE)

6.1.5 GOTO

Aufruf:

```
           +-------+                 +-------+
    +--->! <ZN> !---+     +--->! GOTO !--+
    !      +-------+    V     !       +-------+    V      +-------+
--->+                   +-->+                      +--->! <ZN> !--->
    !                   A     !      +----+        A      +-------+
    +-------------------+     +---->! G !-----+
                                     +----+
```

Funktion:

Mit GOTO kann eine unbedingte Programmverzweigung vorgenommen werden.

Beispiel:

GOTO 1000

Fehler:

05 Programmzeile mit der gewünschten Nummer existiert nicht.

6.1.6 IF

Aufruf:

```
           +-------+
      +-->! <ZN> !--+
      !    +-------+  V   +-----+   +--------------+   +-------+
--->+                 +-->! IF !-->! <Bedingung> !-->! THEN !--->>
      !                A   +-----+   +--------------+   +-------+
      +---------------+

           +--------------+
      +-->! <Anweisung> !--+
      !    +--------------+  V
>>--+                        +-------------------------------------->+--->
      !      +-------+       A  !                                    A
      +---->! <ZN> !--------+  !                +--------------+   !
             +-------+          !            +-->! <Anweisung> !--+ !
                                !   +-------+ !   +--------------+ V !
                                +-->! ELSE !--+                    +--+
                                    +-------+ !      +-------+     A
                                              +---->! <ZN> !-------+
                                                     +-------+
```

Funktion:

Das IF-Statement erlaubt die Formulierung von bedingten Aktionen. Unter Bedingung ist folgende Anordnung zu verstehen:

a) numerischer Ausdruck
b) Ausdruck1 Relationsoperator Ausdruck2

Als Relationsoperatoren für String- <u>und</u> Zahlenvergeich sind zulässig:

>	größer
<	kleiner
=> , >=	größer und gleich
=< , <=	kleiner und gleich
<> , ><	ungleich
=	gleich

Im Fall a): Falls der Wert des numerischen Ausdrucks Null ist, gilt die Bedingung nicht erfüllt, sonst als erfüllt.

Falls die Bedingung erfüllt ist, wird der THEN-Teil ausgeführt. Falls die Bedingung nicht erfüllt ist, wird das nächste Statement im Programm ausgeführt oder die im ELSE-Zweig vorgesehene Aktion.

Beispiel:

```
a) 1000 IF A<>0 THEN 2000
        entspricht
   1000 IF A THEN 2000
b) 1010 IF A<-8 THEN 2020 ELSE A=-100
c) 1020 IF A$<>CHR$(12)+B$ THEN PRINT "UNGLEICH"
d) 1030 IF A>B THEN IF A>C THEN 3000
```

String-Vergleich

Beim Vergleich zweier Strings bedeutet die "größer"- Aussage, daß der ASCII-Code als Hexzahl betrachtet größer ist. Ist der kleinere String identisch mit dem linken Teilstring des Vergleichsobjekts, dann gilt der längere String als größer.

Es gilt z.B.:

```
      "Z"  >  "A"
      "b"  >  "Cxxxx"
   "ABCD"  <  "ABCDEFG"
```

Fehler:

08 Inkompatible Datentypen in den Vergleichsausdrücken

14 Unerlaubte Relation im Bedingungsausdruck

6.1.7 STOP

Aufruf:

```
         +-------+            +-------+
      +-->! <ZN> !--+      +-->! STOP !--+
      !  +-------+  V      !  +-------+  V
 --->+              +--->+               +--->
      !             A      !   +-----+   A
      +-------------+      +--->! ST !---+
                               +-----+
```

Funktion:

Das STOP-Statement legt Abbruchstellen innerhalb eines Programms fest. Bei der Ausführung kehrt der Interpreter grundsätzlich in die Kommandobetriebsart zurück, auch wenn der Abbruch innerhalb eines BASIC-Unterprogramms erfolgte.

Beispiel:

```
1000 IF A=0 THEN STOP
```

Fehler:

(keine)

6.1.8 END

Aufruf:

```
     +-------+    +------+
---->! <ZN> !--->! END !--->
     +-------+    +------+
```

Funktion:

Das END-Statement stellt das letzte Statement eines BASIC-Programms dar. Es ist allerdings nicht notwendig, wenn das Programm über ein STOP oder RETURN-Statement beendet wird

Beispiel:

```
31000  END
```

Fehler:

03 Hinter END existiert eine weitere Programmzeile

6.2 RECHEN-/SPEICHERANWEISUNGEN

6.2.1 DIM

Aufruf:

```
        +------+
    +-->! <ZN> !--+
    !   +------+  V    +-----+
--->+             +--->! DIM !--->>
    !             A    +-----+
    +-------------+
```

```
                          +-----------+
       +--------+  +---+  ! <Ausdruck !                         +---+
>>-+-->! <Name> !->! ( !->!     1   > !-->+---------------->+-->! ) !--+--->
   A   +--------+  +---+  +-----------+   !   +-----------+ A   +---+  !
   !                                      !   ! <Ausdruck ! !          !
   !                                      +-->!     2   > !-+          !
   !                         +---+            +-----------+            !
   +-------------------------! , !<------------------------------------+
                             +---+
```

Funktion:

Mit der DIM-Anweisung können ein- oder zweidimensionale Felder vom Datentyp "REAL" oder "STRING" angelegt werden. Bei Vektoren legt der Wert von "Ausdruck1" die Länge (Anzahl der Elemente) fest. Die Indizierung geht hierbei von 0 bis zur angegebenen Dimensionszahl. Eindimensionale Felder werden grundsätzlich als Spaltenvektoren angelegt, was bei Anwendung des Matrix-Vektor-Kalküls (s. MAT) zum Tragen kommt.

Bei Matrizen legt der Wert von "Ausdruck1" die Anzahl der Zeilen fest, während sich die Anzahl der Spalten aus dem Wert von "Ausdruck2" ergibt. Auch hier beginnt die Indizierung bei 0, wobei allerdings im Rahmen des MAT-Statements nur die Teilmatrix ab Element 1,1 benutzt wird. BASIC legt die Felder zeilenweise ab, d.h. zuerst wird die "nullte"-Zeile, dann die erste usf. abgelegt.

Bei der erstmaligen Anlage von Feldern werden alle Elemente gelöscht (Wert 0 bei REAL-, Leertext bei STRING-Feldern).

Sofern der verfügbare Arbeitsspeicher keine physikalische Begrenzung darstellt, beträgt die maximale logische Dimension

- numerische Vektoren: 8190
- numerische Matrizen: 126 mal 126
- Stringfelder: 255 Elemente

Mit der DIM-Anweisung können bereits vorhandene Felder redimensioniert werden. Hierbei darf das redimensionierte Feld jedoch nicht größer werden, als bei der erstmaligen Anlage angegeben wurde. Die Umwandlung von Vektoren in Matrizen und umgekehrt ist möglich. Bei der Redimensionierung werden die Feldelemente NICHT gelöscht.

Beispiel:

```
1000  DIM   X1(10),A(N,N),Xa$(2,10)
1010  INPUT N
1020  DIM X1(N)
```

Fehler:

13   Dimension zu groß
33   Redimensionierung nicht möglich, Originalfeld zu klein

6.2.2 LET

Aufruf:

```
                          +-----+
        +------+      +-->! LET !--+
    +-->! <ZN> !--+   !   +-----+  !
    !   +------+  V   !            V
--->+             +-->+----------->+---->>
    !             A   !            A
    +-------------+   !    +---+   !
                      +--->! L !---+
                           +---+
```

```
            +------------+
        +-->! <Variable> !----+
        !   +------------+    !
        !                     V
        +-------------------->+
        !                     V    +---+   +-------------+
  >>-+--+                     +--->! = !-->! <Ausdruck>  !--+--->
     A  !   +-----------+     A    +---+   +-------------+  !
     !  !   ! <Ausgabe- !     !                             !
     !  +-->! funktion> !-----+           +---+             !
     !      +-----------+              +--! : !--+          !
     !                                 !  +---+  !          !
     !                                 !         !          !
     !                                 V  +---+  !          !
     +---------------------------------+<-! , !<-+<---------+
                                          +---+
```

Funktion:

Die LET-Anweisung erlaubt die Formulierung von Wertzuweisungen und stellt damit das zentrale Statement dar. Die Ergebnisse von numerischen oder String-Operationen können Variablen des gleichen Typs zugewiesen werden. Numerische Werte können darüberhinaus auch an Spezialfunktionen wie

CNT	Zähler
DAC	Analogausgabe
DOUT	Binärausgabe
PUT	Byteausgabe
COMMAND	Kommandowort-Ausgabe an seriellen Kanal
TUP/TDOWN	(Realitiv-) Zeitgeber
TIME$	Tageszeitgeber) wenn implementiert!
DATE$	Datumsgeber)

übergeben werden.

Beispiel:

```
1000  A_min = B_max, C=A+B^(X+LN(X2*X3)), A$="AB"
1020  FF$=CHR$(12) : B$=FF$+"ABCDE"+CHR$(27)
1030  LET  DAC(N)=K*Y1, DOUT(2)=1, PUT(128)=127
1040  TIME$="08:21:35" : DATE$="22:06:85"
```

Hinweise:

Zur Vermeidung von Mehrdeutigkeiten bei der Eingabe des Schlüsselwortes L(ET) sollte nach dem Schlüsselwort grundsätzlich ein Leerzeichen folgen:

Beispiele:

Eingabe:	Wirkung:
LA = 1	LA =1
L A= 1	LET A =1
LETA=1	LETA =1

Im Zweifel die Zeile auslisten, ob Variablenname bzw. Kurzform richtig erkannt wurde!

Fehler:

07	Anweisung nicht korrekt abgeschlossen
08	Inkompatible Datentypen
10	Ausgabefunktion auf rechter Seite oder: Einlesefunktion auf linker Seite

6.2.3 MAT

Aufruf:

```
        +------+
    +-->! <ZN> !--+
    !   +------+  V   +-----+  +------------+  +---+
--->+             +-->! MAT !-->! <Feldname> !-->! = !--->>
    !             A   +-----+  +------------+  +---+
    +-------------+


       +-----+
   +-->! IDN !-----+
   !   +-----+     !
   !   +-----+     V
   +-->! ZER !-----+-----------------------+
   !   +-----+     A                       !
   !   +-----+     !                       !
   +-->! TRO !-----+                       !
   !   +-----+                             !
   !   +-------------------+               V
>>-+-->! <Matrix-Ausdruck> !-------------->+---->
   !   +-------------------+               A
   !   +-----+   +---------------------+   !
   +-->! TRN !---! (<Matrix-Ausdruck>) !---+
       +-----+   +---------------------+


                      +---+  +-------------------+  +---+
                  +-->! ( !--! <Matrix-Ausdruck> !--! ) !---+
                  !   +---+  +-------------------+  +---+   !
                  !     +------------+                      V
Matrix-   --->+---+-----! <Feldname> !--------------------->+--+---->
Ausdruck:     A   !     +------------+                      A  !
              !   !   +---+  +----------+  +---+            !  !
              !   +-->! [ !--! <Skalar> !--! ] !------------+  !
              !       +---+  +----------+  +---+               !
              !                                                !
              !             +---+                              !
              !         +---! + !<--+                          !
              !         !   +---+   !                          !
              !         V   +---+   !                          !
              +-------->+<--! - !<--+<-------------------------+
                        A   +---+   !
                        !   +---+   !
                        +---! * !<--+
                        !   +---+   !
                        !   +---+   !
                        +---! . !<--+
                            +---+
```

Funktion:

Das Matrix-Statement MAT erlaubt die direkte Berechnung von MatrixVektor-Ausdrücken und stellt damit das Pendant zum LET-Statement für skalare Ausdrücke dar.

Neben der sehr kompakten Formulierung von Matrix-Ausdrücken liegt der Hauptvorteil des MAT-Statements in der deutlich höheren Rechengeschwindigkeit gegenüber der Programmierung "zu Fuß". Die Geschwindigkeitssteigerung liegt zwischen Faktor 10 und 30 und erklärt sich vornehmlich damit, daß für das Aufbrechen des Matrixausdrucks im Vergleich zu den unabdingbaren Rechenoperationen wenig Zeit aufgewendet und keine explizite Indexrechnung gemacht wird.

Damit empfiehlt es sich für den Anwender, seine Probleme zu vektorisieren, um eine Beschleunigung des Rechengangs zu erreichen.

Bei der Bearbeitung des MAT-Statements wird zunächst die rechte Seite unter Berücksichtigung der Syntaxregeln und der Dimensionalität der beteiligten Operanden ausgewertet. Anfallende Zwischenergebnisse werden selbständig angelegt. Das auf der linken Seite stehende Ergebnisfeld wird entsprechend redimensioniert, es sei denn, daß auf der rechten Seite die Sonderfunktionen IDN, ZER oder TRO stehen. In diesen Fällen behält die Ergebnismatrix ihre ursprüngliche Dimensionierung, da diese Funktionen nur eine Wertsetzung vornehmen. Die Zielmatrix darf bei Multiplikation nicht auf rechter Seite stehen.

Im Rahmen eines komplexen Matrizen-Ausdrucks können folgende Operationen und Funktionen vorgenommen werden:

Addition und Subtraktion:

- Matrix/Matrix
- Matrix/Skalar
- Skalar/Matrix
- Vektor/Vektor
- Vektor/Skalar
- Skalar/Vektor

Multiplikation:

- Matrix/Matrix
- Matrix/Spaltenvektor
- Matrix/Skalar
- Skalar/Matrix
- Zeilenvektor/Matrix
- Zeilenvektor/Spaltenvektor (Skalarprodukt)
- Spaltenvektor/Zeilenvektor (Dyad.Produkt)
- Vektor/Skalar
- Skalar/Vektor
- Element - Produkt

Hinweise:
* Beim Skalarprodukt wird das Ergebnis im Element 1,1 einer 1*1-Matrix abgelegt.
* Beim Elementprodukt werden Felder elementweise multipliziert, was insbesondere bei der Normierung/Skalierung von Meßwerten einen häufigen Rechengang darstellt, z.B.:

```
1020   XE_nor = SKV.XE + X_off
```

Matrix-Vektor-Funktionen

TRN	Transposition
IDN	Die Diagonalelemente der Zielmatrix erhalten den Wert 1,alle anderen Elemente den Wert 0.
ZER	Alle Elemente der Zielmatrix werden zu null gesetzt
TRO	verschiebt die Elemente des Zielvektors (!) um eine Indexstelle in Richtung nulltes Element

Beispiel:

```
1000 REM Zustandsdgl. Eingrößensystem
1010   MAT  XP = A*X + B*[U]
1020   MAT Y=CT*X
1030   Y=Y(1,1)+D*U

2000 REM Mehrgrößensystem (zeitdiskret)
2005   VEC U=MADC(MO)
2010   MAT X1= P*X + Q*U
2020   MAT Y=CT*X + D*U
2025   VEC MDAC(SO)=Y
2030   MAT X=X1
```

Weitere Beispiele siehe Programm MATDEM.BAS !

Fehler:

12	Undefiniertes Feld
11	Feldtyp falsch (z.B. ein- statt zweidimensional, String statt Vektor)
30	Unerlaubter Operand im MAT-Statement
31	Falsche Reihenfolge von MAT-Operationen
32	Inkonsistente Dimensionen auf der rechten Seite
33	Zielfeld zu klein dimensioniert
34	Vektor zu groß (>126) für MAT-Operationen
35	Zielfeld auch auf rechter Seite des MAT-Statements
36	Zuwenig Speicher für Ablage der Zwischenergebnisse
37	Interner Konsistenzfehler (Entschuldigung !)

6.2.4 VEC

Aufruf:

```
            +------+
        +-->! <ZN> !--+
        !   +------+  V    +-----+
    --->+             +--->! VEC !--->>
        !             A    +-----+
        +-------------+
        +--------------+                          +--------------+
     +->! <Vektorname> !-----------+           +->! <Vektorname> !--------+
     !  +--------------+           !           !  +--------------+        !
     !  +------+                   V  +---+    !  +------+                V
>>-+-+->! MDAC !--+                +->! = !->+-+->! MDAC !-+              +-+-+-->
   A !  +------+  V  +-----------+ A  +---+  A !  +------+ V  +---------+ A ! !
   ! !            +->!<Stellort- !-+         ! !           +->!<Meßort- !-+ ! !
   ! !  +-------+ A  !   Vektor> !           ! !  +------+ A  ! Vektor> !   ! !
   ! +->! MDOUT !-+  +-----------+           ! +->! MDIN !-+  +---------+   ! !
   !    +-------+                            !    +------+                  ! !
   !                                         !               +---+          ! !
   !                                         !             +-! + !<-+       ! !
   !                                         !             V +---+  !       ! !
   !                                         +-------------+        +<------+ !
   !                                                       A +---+  !         !
   !                                                       +-! - !<-+         !
   !                        +---+                            +---+            !
   +------------------------! , !<--------------------------------------------+
                            +---+
```

Funktion:

Die VEC-Anweisung erlaubt die Formulierung von vektorwertigen Prozeß-E/A-Operationen bzw. PI(D)-Reglerfunktionen. Ähnlich dem MAT-Statement werden durch das Schlüsselwort VEC alle in der Anweisung auftauchenden Variablennamen als Namen eindimensionaler numerischer Felder aufgefaßt (s. Vektor-Funktionen).

Beispiel:

```
1000 REM Einlesen des Meßvektors XE
1010 REM Der Meßortvektor MO enthält in seinen
1020 REM Komponenten die Einlesekanalnummern
1030   VEC  XE=MADC(MO)

2000 REM Ausgabe des analogen Stellvektors Y
2010 REM Der Stellortvektor SO enthält in seinen
2020 REM Komponenten die Ausgabekanalnummern
2030   VEC  MDAC(SO) = Y

3000  VEC Y=PI(Y,E,E1,KP,KI,YO,YU,YD,AK)
```

Fehler:

24 Unterschiedliche Dimension zweier Vektoren

25 (Hardware-)Fehler beim E/A-Zugriff

29 Fehler bei der Bestimmung der logischen Kanaladresse (gewünschter Kanal existiert nicht)

6.3 KOMMENTARANWEISUNGEN

6.3.1 REM

Aufruf:

```
        +-------+            +------+
    +-->! <ZN> !--+      +--->! REM !--+
    !   +-------+  V      !   +------+  V
--->+              +--->+               +--->
    !              A      !    +---+    A
    +--------------+      +---->! R !---+
                                 +---+
```

Funktion:

Das REM-Statement erlaubt die Einfügung von Kommentaren in das Programm. Es hat auf die Programmausführung keinerlei Einfluß.

Beispiel:

```
10  REM Nach REM können beliebige Zeichen folgen.
```

6.3.2 PUNKTBEFEHLE

Aufruf

```
      +-------+       +---+      +---------+
---->! <ZN> !----->! . !---->! <Text> !--->
      +-------+       +---+      +---------+
```

Funktion

MSR-Basic läßt ein führendes "."-Zeichen als Kommentar-Einleitung zu. Damit ist es möglich, zur Verbesserung des Listings WordStar-Punktbefehle zur Drucksteuerung einzusetzen (insbesondere .HE, .FO, .CPnn, .PA).

Beim Listen eines Programms auf Datei erzeugt MSR-BASIC aus jeder Punkt-Anwaisung eine zusätzliche Zeile, die dr Programmzeile vorausgeht und diese Zeile ohne Zeilennummer linksbündig darstellt.

Ein derart generiertes Listen kann über WordStar ausgedruckt werden, ohne daß manuelle Modifikationen erforderlich wären.

Beispiel:

QUELL-TEXT

999 Stop

1000 .HE TASK Messwerterfassung
  :

LIST-TEXT

999 Stop

.HE TASK Messwerterfassung
1000 .he TASK Messwerterfassung
  :

6.4 DATEIBEFEHLE

6.4.1 CREATE

Aufruf:

```
           +-------+             +---------+
     +--->! <ZN> !---+       +--->! CREATE !---+
     !     +-------+   V       !    +---------+   V    +--------------+
 --->+                 +--->+                      +--->! <Dateiname> !--->
     !                 A       !    +-----+          A    +--------------+
     +-----------------+       +--->! CR !--------+
                                    +-----+
```

Funktion:

Das CREATE-Kommando legt eine neue Datei an, falls sie nicht bereits vorhanden ist. Der vollständige Dateiname kann als Stringvariable oder als Textkonstante angegeben werden und muß CP/M-Konventionen entsprechen.

Beispiel:

```
10 CREATE "B:DATEI.TST"
oder:
10 create "b:datei.tst"
```

Fehler:

52 ungültiger Dateiname

6.4.2 OPEN

Aufruf:

```
         +-------+            +-------+
     +--->! <ZN> !-->     +--->! OPEN !--+
     !   +-------+  V      !   +-------+  V    +--------------+
 --->+              +--->+                +--->! <Dateiname> !-->>
     !              A      !    +-----+   A    +--------------+
     +--------------+      +---->! OP !---+
                                +-----+

        +---+     +---------------+     +---+     +-------------+
  >>--->! , !--->! <Kanalnummer> !--->! , !--->! <Dateityp> !
        +---+     +---------------+     +---+     +-------------+
```

Funktion:

Das OPEN-Kommando ordnet einer Diskettendatei einen logischen Kanal (10 ... 19) zu, über die dann mit PRINT ON (Kanalnr.) bzw. INPUT ON (Kanalnr.) zugegriffen werden kann.

Dateiname : entspricht CP/M-Konventionen, Extension muß angegeben werden, Stringvariable oder Textkonstante
Kanalnummer: im Bereich von 10-19 (integer), es können 10 LUs gleichzeitig geöffnet sein.
Dateityp : 0 sequentiell write only
1 sequentiell read only

Beachten: Es kann nicht abwechselnd von einer Datei gelesen und auf sie geschrieben werden. Wenn die Zugriffsart wechselt, muß jedesmal vorher ein CLOSE und ein neues OPEN im entsprechenden Mode erfolgen.

Über eine STATUS-Abfrage (Bit 7=1: Kanal offen, siehe Handbuch) kann vom Programm aus der Erfolg/Mißerfolg der OPEN-Anweisung abgefragt und ggfs. durch eine gezielte Fehlerbehandlung abgefangen werden.

Beispiele:

```
a)          OPEN "B:DATEI.TST",15,0
b)   1010   OPEN  DatNam$ , TxtLU,1
     1020   IF BIT (7,STATUS(TxtLU,1)) =0 THEN 2000
       :
     2000   PRINT "Datei " ; DatNam$; "nicht vorhanden"
       :
```

Fehler:

52 ungültiger Dateiname
56 LU existiert nicht
57 LU bereits geöffnet

6.4.3 CLOSE

Aufruf:

```
          +-------+           +--------+
      +--->! <ZN> !---+    +--->! CLOSE !---+
      !    +-------+   V    !    +--------+   V    +-----------------
  --->+                +-->+                 +-->! <Kanalnummer> !--->
      !                A    !    +-----+      A    +-----------------+
      +----------------+    +--->! CL !-------+
                                 +-----+
```

Funktion:

Schließen von Diskettendateien. Notwendig, besonders beim Schreibzugriff, um Dateien für CP/M zu aktualisieren und die restlichen im Zwischenpuffer gespeicherten Daten auf File zu schreiben.

Die CLOSE-Anweisung erlaubt die Angabe mehrerer oder aller zu schließenden Kanäle.

Beispiel:

CLOSE 15

oder: 1000 CLOSE 10,11,12

bzw. 1000 ALL

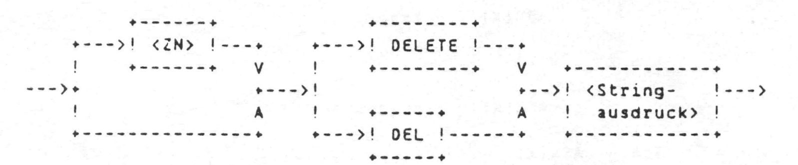
Fehler:

54 Diskette voll

55 Fehler beim Schließen

6.4.4 DELETE

Syntax:



Funktion:

Die durch den Wert des Stringausdrucks bestimmte Datei wird gelöscht. Falls die Datei nicht vorhanden ist, erfolgt keine Fehlermeldung.

Fehlermeldungen:

52 Ungültiger Dateiname

6.5 STANDARDFUNKTIONEN

6.5.1 ABS(X)

ABS(X) Betrag

6.5.2 ACN(X)

ACN(X) Arcuscosinus

6.5.3 ASN(X)

ASN(X) Arcussinus

6.5.4 ATN(X)

ATN(X) Arcustangens

6.5.5 COS(X)

COS(X) Kosinus

6.5.6 EXP(X)

EXP(X) Exponentialfunktion

6.5.7 INT(X)

INT(X) Entier-Funktion

6.5.8 LN(X)

LN(X) natürlicher Logarithmus

6.5.9 LOG(X)

LOG(X) dekadischer Logarithmus

6.5.10 SIN(X)

SIN(X) Sinus

6.5.11 SQR(X)

SQR(X) Quadratwurzel

6.5.12 TAN(X)

TAN(X) Tangens

6.6 STRINGFUNKTIONEN

6.6.1 CHR$(x)

CHR$(x) : Der Wert von x wird als ASCII-Zeichen interpretiert

Beispiel: (Form-Feed, ASCII-Code 0CH)

```
2020 FF$ = CHR$(12)
```

6.6.2 VAL(X$)

VAL(X$) : Die Funktion VAL wandelt ein durch den String X$ gegebene Ziffernfolge in einen Zahlenwert um.

Beispiel:

```
1000  Zahl$ = "123"
1020  PRINT  VAL(Zahl$) AS 10F3
```

liefert:

```
123.000
```

Falls die Ziffernfolge unkorrekt ist, erfolgt die Fehlermeldung ERROR 09.

6.6.3 LEN(X$)

LEN(X$) : Die LEN-Funktion liefert den Wert der Länge des Stringausdrucks.

Beispiel:

```
100 Zahl$="123"
110 PRINT LEN(Zahl$) AS 6I
```

liefert:

```
3
```

Zum Abspalten von Teilstrings aus einem Quellstring stehen in MSRBASIC die bekanntan Funktionen:

6.6.4 LEFT$(Q$,N)

LEFT$(Q$,N): Erzeugt aus den ersten N Zeichen von Q$ einen Teilstring

6.6.5 MID$(Q$,N,M)

MID$(Q$,N,M): Erzeugt einen Teilstring aus dem Quell-string Q$,der beim N-ten Zeichen beginnt und M Zeichen umfaßt

6.6.6 RIGHT$(Q$,N)

RIGHT$(Q$,N): Erzeugt aus den letzten N Zeichen einen Teilstring zur Verfügung.

6.6.7 INSTR({N,}Q$,Z$)

Die INSTR-Funktion prüft einen Quellstring ab einer bestimmten Position auftritt, und liefert im positiven Fall die Position des Suchstrings innerhalb des Quellstrings.

Syntax:

a) INSTR (<Quellstring>,<Suchstring>) : Suche beginnt bei erstem Zeichen des Q$

b) INSTR (<Startposition>,<Quellstring>,
<Suchstring>): Suche beginnt bei der spezifischen Startposition

6.7 SONSTIGES

6.7.1 CALL

Aufruf:

```
        +-------+         +-------+
    +-->! <ZN> !--+   +-->! CALL !--+
    !   +-------+  V   !   +-------+  V   +-----------+
--->!              +->+               +-->! <Nummer> !--->>
    !              A   !   +-----+    A   +-----------+
    +--------------+   +-->! CA !-----+
                           +-----+

          +---+       +--------------+                +---+
   >>--->! ( !-->+----! <Parameter> !------------>+-->! ) !--->
          +---+   A    +--------------+             !   +---+
                  !         +---+                   !
                  +--<-----! , !<------------------+
                            +---+
```

Funktion:

Das Assemblerunterprogramm mit der angegebenen Nummer wird aufgerufen.

Parameter-Übergabe:

Bei Aufruf des Unterprogramms enthält das AL-Register (Z80: A-Register) die Anzahl der übergebenen Parameter, während die Parameter selbst der Reihe nach im Stack übergeben werden (letzter Parameter liegt oben auf dem Stack). Als Parameter-Typen sind Variable und Ausdrücke zugelassen, wobei für jeden Parameter eine 4-Byte-Information übergeben wird, die die Parameter-Adresse (80x86: Offset-Adresse in Bezug auf DS-Register) und einen Typdeskriptor enthält:

```
            MSB                    LSB
+-----------------+-----------------+
! Typ-Deskriptor  !  (reserviert)   !
+-----------------+-----------------+
!         Adresse Parameter i       !
+-----------------+-----------------+
```

Der Typ-Deskriptor ist folgendermaßen aufgebaut:
Bit 7=1: Als Parameter wird ein Ausdruck übergeben
=0: Als Parameter wird eine Variable übergeben

Bit 0=1: Als Parameter wird ein String übergeben
=0: Als Parameter wird eine Gleitkommazahl übergeben

Falls Bit 7=0 (Variablenübergabe):
Bit 1=0: Skalare Variable
=1: Feld(-Element)

Hinweise:

Zur ordnungsgemäßen Rückkehr in den MSR-BASIC-Interpreter kann der RET-Befehl der CPU benutzt werden.

Im UP sllte der Anwender Anzahl und Typ der Parameter überprüfen. Insbesondere ist zu testen, ob der Ergebnisparameter korrekt als Variable übergeben wurde (Beispiel s.u.!).

Die Nummer des Assemblerunterprogramms darf zwischen 0 und 254 liegen. Die Verbindung zwischen Interpreter und Assemblerunterprogrammen erfolgt über eine zu erstellende Sprungtabelle (Adresse CALLTB in Systemkonfigurationstabelle SYSCB, SYSPCxx o.ä.), die folgenden Aufbau hat:

```
CALLTB+
  +0     Anzahl der UPs
  +1     Nummer (in Hex), LSB und MSB der Startadresse des UPs
  +4      dito
```

Nähere Hinweise bitte der Installationsanleitung entnehmen.

Beispiel:

CALL 1(A(0),A$,10*Y)

Im UP kann die syntaktische Korrektheit der Parameter-Übergabe durch folgende Befehlssequenz durchgeführt werden:

```
     UP1:
          CMP    AL,3
          JNZ    ERR23
          POP    AX               ; "10*Y"-Adresse+Typ
          POP    BX
          TEST   AH,00000001B     ; Parameter-Datentyp
          JNZ    ERR01            ; String ist falsch

          POP    AX               ; "A$"-Adresse+Typ
          POP    DX
          TEST   AH,00000001B     ; Parameter-Datentyp
          JZ     ERR01            ; Zahl ist falsch

          POP    AX               ; "A(0)"-Adresse+Typ
          POP    CX
          TEST   AH,10000001B     ; Parameter-Datentyp
          JNZ    ERR01            ; Ausdruck und/oder String
                                    ist falsch
    ; Parameteradressen in CX, DX, BX
            :
          RET

ERR01:    MOV  AL,1
          JP   TERROR      ; MSR-BASIC-Fehlerneahnadlung
```

4380G

Fehler:

01	Falscher Parameter-Typ (z.B. String statt Zahl)
15	Unterprogramm mit der gewünschten Nummer ist nicht vorhanden
23	Parameter-Anzahl nicht korrekt

7. KOMMANDOS

7.1 SYSTEM

Aufruf:

```
        +------+
    --->! SYS !--->
        +------+
```

Funktion:

Verlassen des Interpreters
Rückkehr in das Betriebssystem

Hinweise:

Beim Verlassen des Interpreters wird das MSRBASIC-interne Echtze betriebssystem abgeschalten und das Interruptsystem des Betriebssystems restauriert.

7.2 FREEZE

Aufruf:

```
        +------+
    --->! FRE !--->
        +------+
```

Funktion:

Entfernung gelöschter Programmteile aus Speicher ("Entrümpelung")
Schutz des BASIC-Programms ("einfrieren")
Ankopplung eines in PROMS abgespeicherten BASIC-Programmes
Freigabe des Echtzeitbetriebs

Hinweise:

Das FREEZE-Kommandos gibt die Echtzeitbetriebsart frei (Ausführung von ACTIVATE-TASK-Anweisungen etc. zulässig), was aber aus Sicherheitsgründen ein Verbot des gleichzeitigen Editierens von Programmzeilen nach sich zieht.

7.3 LIST

Aufruf:

```
                              +----------------+
                         +--->! "<Dateiname>" !-----+
      +-------+          !    +----------------+    V
 --->! LIST !--->!------------------------------>+--->>
      +-------+          !    +--------------------+ A
                         +--->! ON(<Kanalnummer>) !-+
                              +--------------------+


 >>-+----------------------------------------------------->+--->
    !   +--------------+                                    A
    +-->! <Startzeile> !-->+--------------------------------+
        +--------------+   !   +---+                        A
                           +-->! , !-->+--------------------+
                               +---+   !   +------------+   A
                                       +-->! <Endzeile> !--+
                                           +------------+
```

Funktion:

Ausgabe des BASIC-Programms auf Bedienkonsole, E/A-Kanal n (z.B. Drucker) oder Datei, wobei eine Startzeile und Endzeile spezifiziert werden kann. Beim Aufruf "LIST ON ..." geht der Interpreter in die Transparentbetriebsart (s. LOAD-Kommando) um die Verbindung zu einem Hostrechner zu ermöglichen ("Virtuelles Terminal"). Diese Betriebsart wird bei Eingabe von ^A verlassen und das Auslisten des Programms auf den angewählten Kanal vorgenommen.

Beim Auslisten wird versucht, die Struktur des Anwenderprogramms möglichst deutlich wiederzugeben. Daher werden Zeilen, die die Befehle

END, REM, RETURN, STOP, WAIT

enthalten, um zwei Druckpositionen nach links gerückt und das Innere von Laufanweisungen um 4 Leerzeichen nach rechts verschoben. Grundsätzlich wird die Normalform eines Statements ausgedruckt, auch wenn der Benutzer ursprünglich die Kurzform eingegeben hat.

Beispiel:

```
1000   REM Demonstrationsprogramm
1010     DIM X (10,10)
1020     FOR I=1 TO 10
1030         FOR J=1 TO 10
1040             PRINT "Element";I AS 2I;",",J AS 2I
1050             INPUT X(I,J)
1060         NEXT J
1070     NEXT I
1080   STOP
```

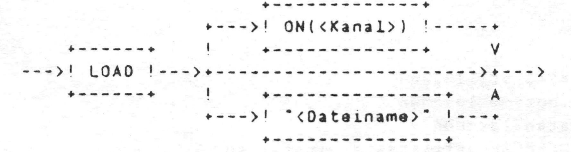
Hinweise: siehe auch Abschnitt "Punktbefehle" und
Abschnitt "Editor-Funktionen"

<u>Hinweise:</u>

In einer MS-DOS- bzw. CP/M- Umgebung kann das Programm auch auf eine Floppy-Disk-Datei ausgegeben werden. Der Dateiname muß DOS- bzw. CP/M-Konventionen entsprechen (Kleinbuchstaben werden als Großbuchstaben betrachtet) und darf keine Extension aufweisen, da die Datei automatisch zu "Dateiname.BAS" angelegt wird. Eine evtl. bereits vorhandene Datai gleichen Namens wird in "Dateiname.BAC" umbenannt (automatisches Back-Up).

7.4 LOAD

Aufruf:



Funktion:

Einlesen eines BASIC-Programmes entweder
- von Bedienkonsole
- von beliebigem Peripheriegerät auf Kanal n
- von Floppy-Disk (nur bei CP/M)

und ablegen in Arbeitsspeicher.

Zu Beginn des Ladens erscheint als Triggersymbol ein !-Zeichens auf der Konsole. Nach dem Einlesen einer Programmzeile wird eine Zifferanzeige am Bildschirm um 1 erhöht (modulo-8-Zähler). Nach dem Einlesen der letzten Programmzeile bleibt die Anzeige stehen, bis nach dem vollständigen Übersetzen des Programmklartexts in den MSR-BASIC-internen Zwischencode die READY-Meldung kommt.

Falls der "ON(...)"-Zweig aktiviert ist, geht der Interpreter zunächst in eine Transparentbetriebsart, d.h. er stellt softwaremäßig die Verbindung zwischen der Bedienkonsole und dem angewählten Peripheriegerät her. Ist dies ein (Host-)Rechner, kann er in dieser Phase des LOAD-Kommandos über das Terminal des MSRBASIC-Rechners nahezu beliebig bedient werden ("Virtuelles Terminal"). Die Eingabe von ^A veranlaßt den MSRBASIC-Interpreter, an den Hostrechner ein RETURN-Zeichen zu schicken und damit das Zusenden eines Programms auszulösen. MSRBASIC geht unmittelbar danach in die Ladebetriebsart über. Das Laden wird abgebrochen, wenn ein ^Z-Zeichen eintrifft oder wenn mehr als 5s kein Zeichen mehr kommt (Time-out).

Das Laden kann von der Konsole über ^C abgebrochen werden, ohne daß bereits eingelesene Programmzeilen verloren gehen.

Hinweise:

Unter CP/M 2.2 ist die Betriebsart als virtuelles Terminal nur nach Installation entsprechender Kanaltreiber möglich (s. Installationsanleitung), unter CP/M plus (3.0) eignet sich hierfür die AUX-Schnittstelle.

Werden beim Übersetzen in den Zwischencode Syntaxfehler erkannt, so wird die fehlerhafte Zeile gemeldet. Alle bereits eingegliederten Zeilen bleiben erhalten, alle folgenden Zeilen gehen verloren.

7.5 NEW

```
                       +------+
Aufruf:           --->! NEW !--->
                       +------+
```

Funktion:

TASKs stillegen
Programm löschen
Daten löschen
NOFREEZE-Betriebsart einstellen

7.6 NOFREEZE

```
                        +-------+
Aufruf:            --->! NOF   !---->
                        +-------+
```

Funktion:

TASKs stillegen
BASIC-Prom abkoppeln
Programmschutz aufheben

7.7 MFREE

```
                      +---------+
Aufruf:         -->! MFREE   !---->
                      +---------+
```

Funktion:

a) in NOFREEZE-Betriebsart
Speicherentrümpelung
Löschen des Datenspeichers
Angabe des freien Speichers

b) in FREEZE-Betriebsart
Angabe des freien Datenspeichers

7.8 RUN

Aufruf:

```
         +------+
     --->! RUN !--->+-----------------------+--->
         +------+   !     +---------------+  A
                    +--->! <Startzeile> !---+
                          +---------------+
```

Funktion:

Starten eines BASIC-Programms bei der ersten Zeile oder der Startzeile. Die Symboltabelle wird in beiden Fällen nicht gelöscht.

7.9 SAVE

Aufruf:

```
                           +-----------------+
                      +--->! "<Dateiname>" !--------+
       +-------+      !    +-----------------+      V
   --->! SAVE !--->!----------------------------->+--->>
       +-------+      !    +---------------------+  A
                      +--->! ON(<Kanalnummer>) !---+
                           +---------------------+

   >>-+--------------------------------------------------------+--->
      !  +---------------+                                      A
      +-->! <Startzeile> !-->+----------------------------------+
         +---------------+   !   +----+                         A
                             +-->! , !-->+----------------------+
                                 +----+  !   +--------------+   A
                                         +-->! <Endzeile> !--+
                                             +--------------+
```

Funktion:

Ausgabe des im Speicher befindlichen BASIC-Programms auf Bedienkonsole oder Peripheriegerät, ähnlich wie LIST-Kommando. Im Gegensatz zum LIST-Kommando wird das Listing nicht strukturiert.

Beispiel:

```
1000 R Demonstrationsprogramm
1010 DIM X(10,10)
1020 F I=1 TO 10
1030 F J=1 TO 10
1040 P "Element"; I AS 2I;",";J AS 2I
1050 I X (I, J)
1060 N J
1070 N I
1080 STOP
```

7.10 VCL

Aufruf:

```
            +------+
  ---->! VCL !--->
            +------+
```

Funktion:

- Löschen aller Anwendervariablen und -Zeitgeber

Das VCL-Kommando ist aus Sicherheitsgründen nur in der NOFREEZE-Betriebsart (kein Realzeitbetrieb) zulässig.

8. KOMMUNIKATION

8.1 ÜBERSICHT

8.1.1 AUSGANGSSITUATION

Die Kommunikationsfähigkeit von Automatisierungsgeräten untereinander ist angesichts der CIM-Zielsetzung ein Gebot der Stunde.

MSR-BASIC hat von Anfang ein sehr offenes Kommunikationskonzept mit

- freier Wahl und Konfigurierung der Schnittstellen,
- leistungsfähigen Kanaltreibern auf Interruptbasis,
- integriertem Terminal-Modus
- integrierten Up-/Download-Fähigkeiten und
- leistungsfähigen Sprachmitteln wie
  "PRINT ON"-, "INPUT ON"-Anweisung und
  "INCHAR$"-, "STATUS"- und "COMMAND"-Funktionen,

zur Verfügung gestellt.

8.1.2 ERWEITERTE KOMMUNIKATIONSFUNKTIONEN

Ab Revision 3.1. (Sommer 87) werden diese Fähigkeiten ausgebaut durch das Kommunikations-Paket MSRCOM (als eine Standard-Option) für MSR-BASIC-Systeme, das den Aufbau von MSR-BASIC-Netzwerken durch folgende High-Level-Funktionen unterstützt:

- Zugriff auf externe Prozeßkanäle (Remote I/O-Access)
- Zugriff auf externe Variable
- Protokollverwaltung

Diese Funktionen sind vor allem für jene Einsatzfälle hervorragend geeignet, bei denen eine Zentralstation auf verteilte Vor-Ort-MSR-Rechner bidrektional zugreifen soll.

Das Kommunikationspaket MSRCOM ist in Anlehnung an das OSI-ISO-Schichtenmodell für offene Kommunikation streng modular ausgelegt.

D.h. daß der Austausch einer unteren (Software-)Schicht die oberen Schichten nicht berührt.

Durch die Bereitstellung geeigneter Kanal-Treiber für serielle Schnittstellen, wird die Vernetzung über diese preisgünstigen Standardschnittstellen besonders unterstützt.

Im Sinne der Offenheit zu anderen Hard- und Software-Welten beinhaltet MSRCOM ein Turbo-PASCAL-Paket MSRLINK, mit dem der Einsatz eines Standard-PCs als Master unterstützt wird (siehe separate Beschreibung).

8.1.3 KONFIGURATIONEN

MSRCOM unterstützt prinzipiell folgende Konfigurationen:

a) 1-Master-1-Slave-Konfiguration (Bild 1)
b) 1-Master-n-Slave-Stern-Konfiguration (Bild 2)
c) 1-Master-n-Slave-Bus-Konfiguration (Bild 3)

Der Konfigurationstyp bestimmt den Umfang und die Art der MSRCOM-Module, die im konkreten Fall eingesetzt werden (s. Abschnitt Installation/Konfiguration).

```
+------------+        +------------+        +-----------+
! MSR-BASIC- !<------>! MSR-BASIC- !<- - ->! Techn.    !
! Master     !        ! Slave      !        ! Prozeß    !
+------------+        +------------+        +-----------+
```

Schaubild 8-1: 1-Master-1-Slave-Konfiguration

```
                         +------------+
                         ! MSR-BASIC- !
                         ! Master     !
                         +------------+
                           ! ! ...  !
  +-------------------+    !        +------------------+
  !                        !                           !
  V                        V                           V
+------------+       +------------+             +------------+
! MSR-BASIC- !       ! MSR-BASIC- !             ! MSR-BASIC- !
! Slave 1    !       ! Slave 2    !   . . . .   ! Slave n    !
+------------+       +------------+             +------------+
    !     !             !     !                    !     !
+------------+       +------------+             +------------+
!  Techn.    !       !  Techn.    !             !  Techn.    !
!  Prozeß    !       !  Prozeß    !   . . . .   !  Prozeß    !
+------------+       +------------+             +------------+
```

Schaubild 8-2: Stern-Konfiguration

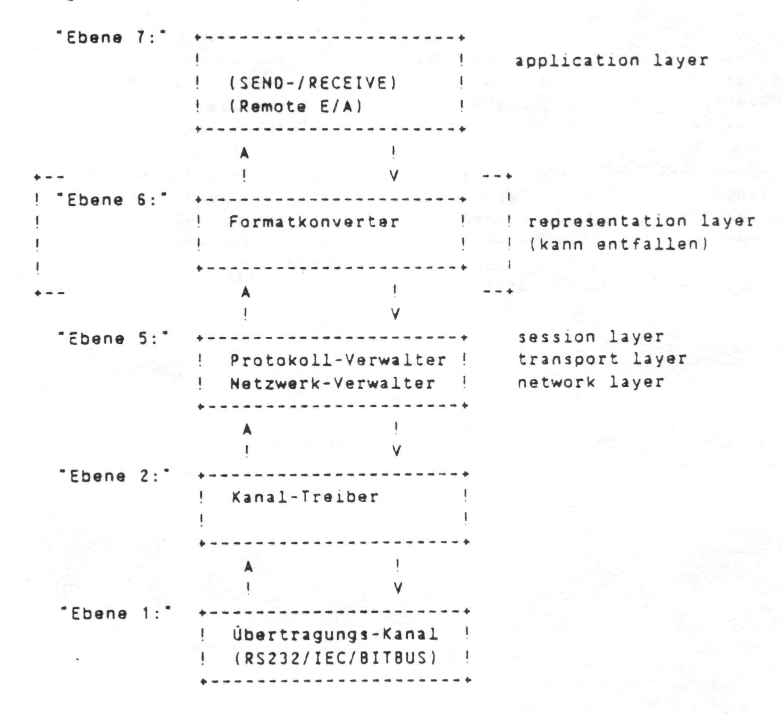
```
                    +-------------+
                    ! MSR-BASIC-  !
                    ! Master      !
                    +-------------+
                           !
    +----------------------+----------------------+
    !                      !                      !
    V                      V                      V
+-------------+     +-------------+        +-------------+
! MSR-BASIC-  !     ! MSR-BASIC-  !        ! MSR-BASIC-  !
! Slave 1     !     ! Slave 2     !        ! Slave n     !
+-------------+     +-------------+        +-------------+
     !     !             !     !                !     !
 +-----------+       +-----------+          +-----------+
 !  Techn.   !       !  Techn.   !          !  Techn.   !
 !  Prozeß   !       !  Prozeß   !          !  Prozeß   !
 +-----------+       +-----------+          +-----------+
```

Schaubild 8-3: Bus-Konfiguration

8.2 KOMMUNIKATIONS-MODELL

Die Kommunikation in MSR-BASIC erfolgt noch einem vereinfachten Schichten-Modell im Sinne des ISO-OSI-Standards.

Folgende Ebenen sind implementiert bzw. werden unterstützt.



8.3 ANWENDER-EBENE

8.3.1 MASTERFUNKTIONEN

Im MSRBASIC-Master stehen dem Anwender für die Kommunikation die

- Variablen-Zugriffsfunktionen "SEND" und "RECEIVE",
- E/A-Zugriffsfunktion "ADC", "DAC", "DIN", "DOUT", "CNT" bzw. "RADC", "RDAC", "RDIN", "RDOUT", "RCNT"

zur Verfügung.

8.3.1.1 RECEIVE-FUNKTION

Ihre Syntax lautet:

<Ziel-Variable> = RECEIVE(<Quellbezeichner>,<Steuerwort>,<Statuswort>)

Bedeutung der einzelnen Parameter:

Quellbezeichner : Stringausdruck, der den Namen jener Slave-Variablen liefert, deren Momentanwert angefordert und an die Zielvariable übertragen werden soll.

Der Name einer Slavegröße setzt sich zusammen aus:

- Stations-Nr. des Slaves (ggfs.)
- lokaler Variablenname

Syntaxdiagramm:

```
     +-----------+
     ! Stations-!
  +->! Nummer    !--+
  !  !           !  !
  !  +-----------+  !
  !                 V     +----------------+
--+-----------------+---->! Variablen-Name !-->
                          +----------------+
```

Steuerwort: Numerische Größe, die auf 8 Bit gerundet wird. Die letzten vier Bits legen die Slaveadresse fest (in der Regel Adresse 0).
Ist das 8.Bit gesetzt (Steuerwort > 127), werden zum Zweck der Fehlersuche alle Bytes im Hex-Format angezeigt, die zwischen dem MSRBASIC-Master und dem Slave ausgetauscht werden.

Außerdem lösen in dieser Betriebsart alle Fehler, die bei der Ausführung von SEND/RECEIVE auftreten, eine MSRBASIC-ERROR-Meldung

ERROR 6x IN LINE nnnn

aus (s.u.!).

Statuswort: Numerische Variable, die die Information über mögliche Fehler bei der Ausführung der SEND/RECEIVE-Funktion wiederspiegelt:

Status-code	ERROR-code	Bedeutung
0	--	Übertragung fehlerfrei
1	61	Der Name einer Slave-Größe ist syntaktisch falsch
2	62	Zeitüberschreitung beim Warten auf Kanalfreigabe
3	63	Zeitüberschreitung beim Senden
4	64	Zeitüberschreitung beim Empfangen
5	65	otschaft syntaktisch falsch (z.B. verstümmelte Zeichen, Prüfsummenfehler)
6	66	Botschaft inhaltlich falsch

NB: Der ERROR-Code wird nur in der Testbetriebsart (Steuerwort >127) angezeigt.

Beispiel:

Im Masterprogramm soll der Inhalt der Variablen "ErrCod$" im MSR-BASIC-Slave Nr. 2 lesen und in seiner lokalen Variablen "ErrNam$" ablegen:

```
VarNam$ = "01:ErrCod$"
ErrNam$ = RECEIVE(VarNam$,0,STWORT)
```

8.3.1.2 SEND-FUNKTION

Der Aufruf der Sendfunktion erfolgt nach der Syntax:

SEND(<Zielbezeichner>,<Steuerwort>,<Statuswort>) = <Ausdruck>

<u>Beispiel:</u>

SEND(VarNam$,CmdByt,StByt) = Quelle

Bedeutung der Parameter:

Zielbezeichner: Name der Slavegröße, die den Wert des Ausdrucks annehmen soll.

8.3.1.3 PROZESSZUGRIFFE

Der Master kann nicht nur auf Variable, sondern auch auf Prozeß-Ein-Ausgabekanäle - lesend - zugreifen.

Hierbei gibt es zwei Varianten:

In der 1-Master-1-Slave-Version besitzt der Master das Prozeßabbild des Slave-Rechners, auf das er mit den üblichen Funktionen wie "ADC", "DIN", "DOUT" etc. zugreift.

In der 1-Master-n-Slave-Version stellt das Master-Kommunikationsmodul spezielle Funktionen wie RADC (remote ADC), RDIN, RCNT, RDOUT, RDAC zur Verfügung, die grundsätzlich 2 Argumente aufweisen:

(<Slave-Nr.>,<Kanal-Nr.>)

8.3.1.3.1 RADC(S,N)

Einlesen eines 16-Bit-Werts vom Analogkanal N des entsprechenden Slave-Rechners S,
Skalierung ist konfigurationsabhängig

8.3.1.3.2 RDIN(S,N)

Einlesen des Binärkanals N des Slave-Rechners S
Wert beträgt 0 oder 1

8.3.1.3.3 RCNT(S,N)

Einlesen eines 16 Bit Zählerkanals N des entsprechenden Slave-Rechners S

8.3.1.3.4 RDOUT(S,N)

Setzen des Binärkanals N des Slave-Rechners S
Wert beträgt 0 oder Nicht-Null

8.3.1.3.5 RDAC(S,N)

Ausgabe auf Stellkanal N (16 Bit genau) des Slave-Rechners S, Skalierung ist konfigurationsabhängig

8.3.1.4 PERIODISCHES UPDATE

Erfolgt ein Master-Prozeßzugriff aus einer TASK oder einer SEQuenz heraus, so sorgt das Kommunikationsmodul dafür, daß periodisch der gesamte E/A-Vektor des betroffenen Slaves angefordert wird.

In dieser Betriebsart tritt keine Verzögerung bei den Master-Zugriffsfunktionen.

Falls mehrere Sekunden kein Anwenderprogrammzugriff auf eine Slaveeinheit erfolgt, wird der periodische Update-Mechanismus für diesen Teilnehmer eingestellt.

8.3.2 SLAVE-FUNKTIONEN

8.3.2.1 MY_ADR

In einer **Sternstruktur** braucht der Slave keine Adreßerkennung durchzuführen, da die Adressierung implizit durch den Master vorgenommen wird. D.h. wenn eine Nachricht eintrifft, dann wird der Master die korrekte LU-Leitung schon gewählt haben.

In einer **Busstruktur** ist hingegen die korrekte Funktion der Adreßerkennung Grundvoraussetzung für den Betrieb des NTS.

Hierbei muß jeder Slave eine eigene Adresse haben. Diese Adresse kann sowohl im Sinne einer Voreinstellung durch Konfiguration des MSR-BASIC-Interpreters festgelegt werden (siehe kapitel "Installation") als auch jederzeit vom Slave-Anwenderprogramm aus abgefragt und modifizert werden.

Hierzu steht die numerische Systemvariable "MY_ADR" zur Verfügung:

```
Stationsadresse "n" setzen:   MY_ADR = n
Stationsadresse abfragen:    P(rint) MY_ADR
```

Über die Systemvariable MY_ADR kann die lokale Teilnehmeradresse eines MSR-BASIC-Slaves gesetzt und abgefragt werden.

Sonderfälle:

> Falls MY_ADR den Wert 0 liefert, ist das betreffende MSR-BASIC-System als Master konfiguriert.
>
> Liefert MY_ADR den Wert -1, dann ist kein Kommunikationsmodul installiert.

8.4 DEMOPROGRAMM RECDEM.BAS

8.4.1 ZWECK

Das mitgelieferte Programm RECDEM.BAS veranschaulicht den wahlfreien Zugriff eines Masterrechners auf beliebige Hostvariable über die RECEIVE-Funktion. Hierbei ist im Slaverechner kein Programm aktiv.

Als Testaufbau kommt die 1-Master-1-Slave-Struktur zum Tragen (Bild).

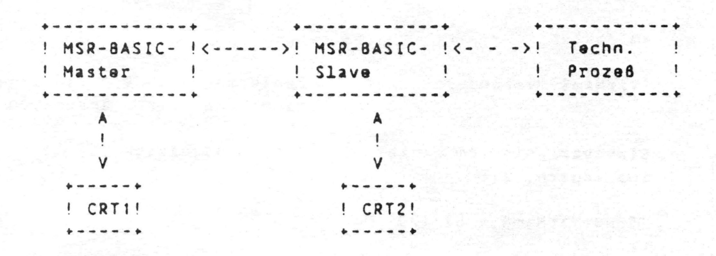


Schaubild 8-4: 1-Master-1-Slave-Konfiguration für RECDEM

8.4.2 FUNKTIONSBESCHREIBUNG

Das Programm RECDEM greift im 1-s Takt (TASK 1) auf eine Slave-Variable zu, deren Namen in der Mastervariablen "VarNam$" enthalten ist.

Voreinstellung fuer den Variablennamen ist "a1".

Der Slave-Variablenname, ihr aktueller Wert und der Status der letzten Übertragung werden in der obersten Zeile angezeigt (Bild).

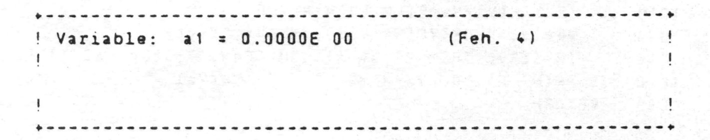


Schaubild 8-5: Bildschirmanzeige im Master

Solange die gewünschte Variable im Slave noch nicht existiert, erscheint eine Fehlermeldung (Feh. 4).

Andernfalls wird ihr aktueller Wert angezeigt.

8.4.3 DEMOEINGRIFFE

Folgende Demoeingriffe sollten durchgeführt werden:

Eingriff	Wirkung
Am Slave:	Master:
a1 manuell verändern	aktuelle Wertanzeige
Am Master:	
"VarNam$" verändern	Fehlermeldung 4, solange Variable nicht existiert, ansonsten Wertanzeige
Slavevariable vom Master aus ändern, z.B.:	Änderungsanzeige
"SEND(VarNam$,a,b)=120"	

8.4.4 LISTING

```
 100   REM RECDEM 07.10.87
 110     VarNam$="a1"
 120     anzeig=1
 130     DEFINE TASK anzeig,1,1000
 140     COMMAND(0)=5*256
 150     PRINT Ä1,0Ü "Slave-Variable: ";
 160     ACTIVATE TASK anzeig
 999   STOP

1000   REM anzeig
1010     IF LEN(VarNam$)=0 THEN RETURN
1030     VarVal=RECEIVE(VarNam$,CmdByt,StByt)
1040     IF StByt THEN PRINT Ä1,50Ü "Feh. ";StByt AS 1I ELSE PRINT Ä1,50Ü "
1050     PRINT Ä1,20Ü VarNam$; " = "; VarVal
1099   RETURN
```

9. PROGRAMMIER-TIPS

MSR-BASIC ist nicht nur ein leistungsfähiges Werkzeug für die Lösung kleinerer MSR-Aufgaben, sondern eignet sich auch für die komplette Automatisierung von Industrieanlagen mit Ein- und Mehrrechner-Systemen.

Um derart komplexe Anwendungen erfolgreich zu meistern, empfiehlt es sich, bei der Systemanalyse, Programmkonzeption und Implementierung einige "goldene Regeln" zu beherzigen, die im folgenden als "Programmiertips" formuliert sind und die Erfahrungen aus zahlreichen Anwendungen wiederspiegeln.

9.1 SYSTEMANALYSE UND PROGRAMM-KONZEPTION

9.1.1 ANALYSE DER ERFORDERLICHEN PARALLELITÄT

Auch wenn es altväterlich klingt: Grundvoraussetzung für ein tragfähiges Echtzeit-Anwenderprogramm ist **eine klare Spezifikation und ein darauf abgestimmtes Programmkonzept**.

Folgende Punkte sollten bei der Programmkonzeption der Reihe nach geklärt werden:

- In welche funktionell entkoppelten Teilbereiche kann die zu automatisierende Anlage zerlegt werden?
- Welche Funktionen sollen/müssen parallel bearbeitet werden?
- Welche Funktionen sind quasikontinuierlich oder ablauforientiert?

Im Prinzip wird jeder parallel ablaufenden Funktion ein MSR-BASIC-Programm vom Typ TASK oder SEQuenz zugeordnet.

Hierbei leistet eine Darstellung der Aufgabe in Form eines Petrinetzes wertvolle Hilfe, da es den Grad an jeweils notwendiger Parallelität strukturell ausweist.

Es empfiehlt sich jedoch im Sinne einer möglichst guten Übersichtlichkeit, die Anzahl an parallelen Programmen nicht über das nötige Maß hinaus anwachsen zu lassen.

Hierzu werden Ablaufaktionen, die zwar parallel laufen können, ohne Verzicht auf die Güte des technischen Prozesses auch nacheinander ausgeführt werden können, als Teilaktionen einer SEQuenz formuliert.

<u>Beispiel:</u> Wegeschaltung in Materialflußsystemen

Bei Materialflußsystemen (z.B. Tanklager) tritt häufig die Aufgabe auf, einen neuen Weg zu schalten. Im konkreten Fall sollen die Ventile V2, V3, V8 geöffnet und die Ventile V1, V5 geschlossen werden, bevor über ein Ventil V10 der Fluß freigegeben wird.

Bei jedem Ventil muß die Endposition überwacht werden, d.h. jeder Vorgang besteht aus den Aktionen:

- Ventil öffen (bzw. schließen)
- max. n Sekunden warten, bis Endposition erreicht, sonst Fehlerbehandlung

Die einzelnen Öffnungs-/Schließungsvorgänge können parallel ablaufen. Dennoch ist es nicht sinnvoll, für jeden Vorgang eine SEQuenz zu "verbrauchen". Vielmehr können alle Aktionen mit **einer** SEQuenz nach folgendem Schema durchgeführt werden:

- Schließe bzw. öffne alle fraglichen Ventile
  (" DOUT(V1)=0, ... DOUT(V2)=1")

- Warte max n Sekunden auf korrekte Positionsrückmeldung:

```
WAIT MAX n FOR NOT(DIN(xV1)+DIN(xV5) * DIN(xV2)... ELSE xxx
```

- Die Fehlerbehandlung in Zeile xxx analysiert die Rückmeldesignale und liefert in der Variablen Err$ die Fehlercodes:

```
IF DIN(xV1)   THEN Err$=Err$+" V1"
IF DIN(xV5)   THEN Err$=Err$+" V5"
IF DIN(xV2)=0 THEN Err$=Err$+" V2"
      :              :
```

REGEL: "soviel Parallelität wie nötig - so wenig wie möglich!"

9.1.2 RESTRIKTIVE AUSLASTUNG

Als weitere Grundregel bei der Konstruktion von Realzeitsystemen sollte man bei der Auslegung der Abtastzeiten eher restriktiv vorgehen, um der Gefahr einer auch nur partiellen Überlastung des Rechners vorzubeugen.

9.2 PROGRAMM-IMPLEMENTIERUNG

9.2.1 ANLAGENORIENTIERTE MODULARISIERUNG

In aller Regel läßt sich eine größere Anlage in definierte Teilbereiche untergliedern. Diese Struktur ist beim Programmentwurf grundsätzlich dadurch zu berücksichtigen, daß

a) alle zum Anlagenabschnitt gehörenden Programmteile (TASKs und SEQuenzen) möglichst einem zusammenhängenden Abschnitt im Gesantprogramm belegen ("Kompaktheit")

b) die Abhängigkeit der abschnittspezifischen Programmteile von den anderen anderen Abschnitten minimiert wird ("Lokalität")

Diesen Zielen kann in MSR-BASIC mit folgenden Mittel Rechnung getragen werden:

- Zeilennummer-Vergabe: Tausender-Gruppen für die Teilbereiche reservieren, z.B.:

```
2000 bis 2999: Anlagenabschnitt 1
3000 bis 3999: Anlagenabschnitt 2
    etc.
```

- Standard-Speicherbelegungs-Struktur:

```
x000    +-----------------+
        ! Initialisierung !

        !                 !
        +-----------------+

        +-----------------+
xy00    ! Sequenz i       !


        +-----------------+
xz00    ! Sequenz i+1     !
```

- Programm-Module logisch verschieblich machen:
  Jeder Programmabschnitt benötigt eine bestimmte Zahl an MSR-BASIC-Betriebsmittel (SEQuenzen, TASKs, TUPs, TDOWNs, FOR/NEXT-Schleifenzähler).

  Mit Hilfe globaler Zustands-Variabler (Vorschlag: "sz", "tz", "tuz" und "tdz") richtet sich bei der Initialisierung jeder Programmabschnitt hinsichtlich seiner SEQuenz-, TASK-, TUP- und TDOWN-Nummernvergabe nach dem vorgegebenen Stand und modifiziert die Zustands-Variablen entsprechend.

Beispiel:

```
2000   REM  Ini-Vorreinigung
2005   REM  2 SEQuenzen: Haupt- und Manual-Betrieb
2010     VRHpt = sz, sz=sz+1
2011     DEFINE SEQ VRHpt,2200
2012     VRman = sz, sz=sz+1
2013     DEFINE SEQ VRman,2800
2020   REM  1 TDOWN-Zeitgeber
2025     t_VR = tdz, tdz=tdz+1
2099   RETURN Ini VR
```

9.3 WEITERE PROGRAMMIERTIPS

- kleinschreibung für Variablennamen verwenden:

Vergleiche:

```
IF HAND + NOTAUS THEN DOUT(HAUPT)=0
IF Hand + Notaus THEN DOUT(Haupt)=0
```

- Sprungziele von GOTO,GOSUB,THEN,ELSE als REM markieren:

```
 980    GOTO 1020

  :
1020  REM *
1030    :
```

Vorteile: Höhere Übersichtlichkeit
Höhere Sicherheit gegen Programmierfehler

- Symbolische Namen für E/A-Kanäle verwenden

Vergleiche:

```
WAIT FOR DIN(123)*DIN(121)+DIN(125)

WAIT FOR DIN(Max1)*DIN(Frei)+DIN(Stop)
```

Vorteile: höhere Selbstdokumentation
Zentrale Änderung durch Neuzuweisung an Variable möglich
Bessere Weiterverwendbarkeit von Programmm

- Symbolische Namen für Logische Einheiten verwenden

```
OPEN  Filnam$,ErrLU,1
IF BIT(FOpen,STATUS(ErrLU))=0 THEN ...
```

- Symbolische Namen für Zeitgeber verwenden:

```
if tdown(t_regl)>0 then return
tdown(t_regl) = T_Regl
```

- Symbolische Namen für TASKs und SEQuenzen verwenden

Vergleiche:

```
IF DIN(3)=0 THEN SUSPEND TASK 2

IF DIN(AStop)=0 THEN SUSPEND TASK Anzeig
```

VORTEILE:

- TASK-/SEQ-Module werden dadurch "verschieblich",d.h. das Verändern der TASK-/SEQ-Priorität wirkt nicht auf die ACTIVATE-SUSPEND-Anweisungen zurück. Aus dem gleichen Grund können Module mit symbolischer "Adressierung" leichter weiterverwertet werden.

- Bildschirm-Steuerzeichen zentral als String-Block definieren:

```
esc$     =CHR$(27)
clreol$  = .....
invon$   = .....
invof$   = .....
```

Vorteil: Hauptteil des Programms bleibt terminal-unabhängig, da Steuersequenzen über ihren Namen ("symbolisch") angesprochen werden.

- Bei Parameterübergabe an Unterprogramme MAT-Anweisung einsetzen:

```
MAT  <arbeitsmatrix>=<Haupt-Matrix>
GOSUB ....
MAT  <Hauptmatrix>  =<Arbeitsmatrix>
```

10. FEHLERMELDUNGEN

00 interner Konsistenzfehler beim Listen
01 inkompatibler Datentyp z.B.: x=a$
02 unerlaubte Kontrollvariable in PRINT- oder INPUT-and-proceed
03 nach END existiert noch eine Programmzeile
04 fehlerhafte Zeilennummer in GOTO etc.
05 gewünschte Zeilennummer existiert nicht
06 ein unerwartetes Zeichen verursachte den Fehler
07 ein Statement wurde nicht beendet
08 ein Statement wurde fehlerhaft formuliert
09 Fehler beim Wandeln einer numerischen Konstanten
10 eine Funktion wurde falsch benutzt (linke statt rechte Seite etc.)
11 Feldzugriff stimmt nicht mit Felddeklaration überein (Dimensionalität, bzw. Datentyp)
12 Zugriff auf Feld vor seiner Definition
13 Fehler bei der Bestimmung einer Zahl - zu groß
14 unerlaubte Relation in einem IF - o. WAIT-Statement
15 Nummer eines Assembler-UPs wurde nicht gefunden

FOR-NEXT-Fehler

16 fehlendes '=' in einem FOR-Statement
17 fehlendes TO oder STEP
18 Ineinanderschachtelung von mehr als 10 Laufvariablen
19 NEXT wurde vor FOR ausgeführt
20 Laufvariable in FOR und NEXT stimmen nicht überein bzw. bereits vorhanden (Multitasking!)
21 Laufvariable ist ein Vektor oder String

22 das gewünschte Vektorelement existiert nicht
23 Anzahl der übergebenen Parameter bei Funktionen mehrerer Variabler stimmt nicht (z.B.:PID)
24 Dimensionen zweier Vektoren stimmen nicht überein
25 Fehler (Hardware) im I/O-Verkehr
26 Task bzw. Ablaufsteuerung sollte aktiviert bzw. desaktiviert werden, bevor sie eingetragen war
27 WAIT wurde als direktes Statement benutzt
28 Zeitüberschreitung beim Warten auf das Eintreffen einer Verriegelungsbedingung
29 Fehler bei der Bestimmung einer logischen Kanaladresse

Matrix-Vektor-Fehler

30 unerlaubter Operand im MAT-Statement
31 falsche Reihenfolge von MAT-Operatoren
32 inkonsistente Dimension auf der rechten Seite
33 Zielfeld zu klein dimensioniert
34 Vektor zu groß (>126) für MAT-Operation
35 Zielfeld auch auf rechter Seite des MAT-Statements
36 zuwenig Speicher für Ablage der Zwischenergebnisse
37 interner Konsistenzfehler (Entschuldigung!)

38 WAIT in GOSUB-Unterprogramm nicht erlaubt
39 (reserviert)
40 Interpreter nicht in Echtzeit-Betriebsart (FREEZE-Kommando eingeben!)
41 INPUT-Anweisung wurde zweimal aufgesetzt (Multitasking)
42 Timeout bei Ausgabe auf seriellem Kanal

Dateiverwaltungs-Fehler (nur in CP/M-Umgebung)

50 unbekannter oder in der vorliegenden Fassung nicht implementierter Befehl (Funktion)
52 Dateiname ungültig
53 Datei nicht vorhanden bzw. Directory-Fehler
54 Diskette voll
55 Fehler beim Schließen einer Datei
56 LU existiert nicht
57 LU, die geöffnet werden soll, ist bereits geöffnet
58 lesen von einer als write-only eröffneten Datei
59 schreiben auf eine als read-only eröffnete Datei
78 interner Konsistenzfehler